自律エージェントとは何か

新出尚之(奈良女子大学)

2016年11月15･16日

本資料は、放送大学奈良学習センターの2016年度面接授業
「自律エージェントとは何か」の講義資料を一部改訂したものです。

はじめに

- 人間と同様に、目的を持ち、その達成のために行動するようなロボットは作れるか?
  → 「自律」エージェント
- そのようなものを作るのに何が必要だろうか?
  - ⋆ 思考能力 (達成の方法を考えるために)
  - ⋆ 身体能力 (実際に動くために)
    - ∗ 今回の話では「思考能力」の方が主役

はじめに(continued)

- 近年、人工知能の進展が著しい
- 分類問題(画像認識、音声認識)や局面評価(将棋、囲碁)の性能は格段に上がった
- しかし、「目的を持ち、達成する方法を考えて行動する」ロボットのために必要な能力は、それらではカバーできない
  - ⋆ 目的を達成する方法(の候補)を求める
  - ⋆ どの方法が現在の条件において使えるかの判断

はじめに(continued)

- そのような能力の実現には「推論」が重要な役割を持つ
  - ⋆ 「推論」には、新たな結論を得る効果の他、得た結論の理由を説明する効果もある
- そこで、今回は論理的な枠組みを(主に)用いて行為の選択を決めるシステムの話

自律エージェント

人間の行為決定

例 出張のため、飛行機に乗るべく空港バスを待っていたが、遅れそうなのでタクシーに切り替え。出発ターミナルが何番か知らなかったので、タクシー内から航空券記載の案内電話に電話して確認。

チェックインには長蛇の列。しかし、預け手荷物がなければエクスプレスチェックインが使えることを思い出し、そちらへ。セキュリティゲートを通ろうとすると引っかかる。携帯電話をポケットから取り出してOK。

出発ラウンジでカフェを探してサンドイッチで朝食。食べながらメールをチェック。出発時間のコールが聞こえたので搭乗口へ。(Bordini et al., *Programming multi-agent systems in AgentSpeak using Jason*, Wiley, 2007. 一部改変)

人間の行為決定(continued)

- 何らかの目的を持ち、その達成のための行動をとる
  - ⋆ 達成方法の案(プラン)を持っている
  - ⋆ プランは部分的にしか具体化されていないが、その場で具体化しつつ実行できる
- 何かうまくいかなければ回復のため、他のプラン(または目的)を選んで実行を試みる
  - ⋆ すぐに諦めたり家に戻ったりはしない
- 実行中に追加の情報が必要なことがある
  - ⋆ それを知るため追加の行為(アクション)を実行する
- それぞれの目標を持った複数の活動を並立させる

コンピュータの“行為”決定

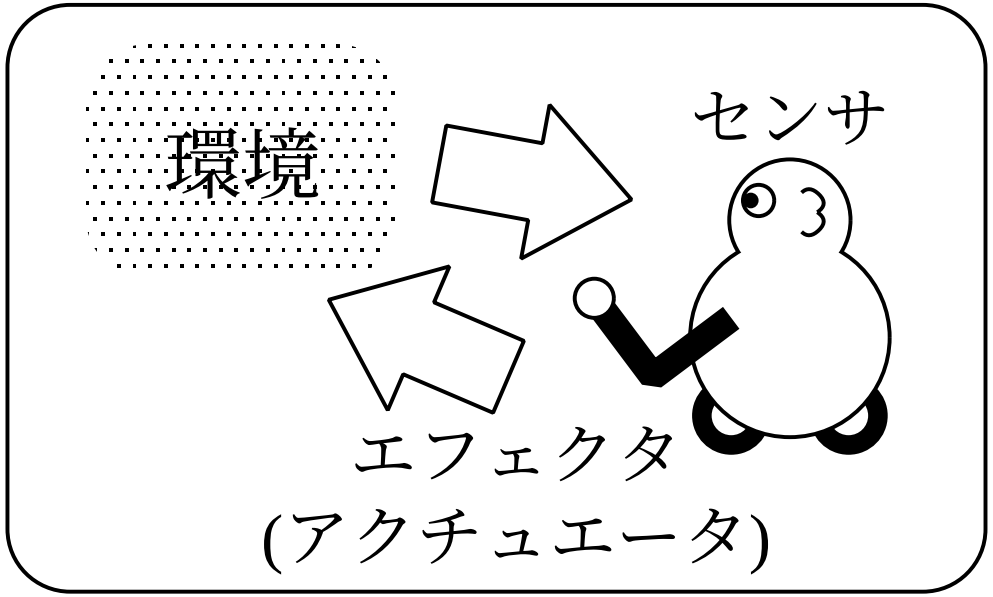
- 多くのコンピュータシステムは、ここに挙げたような性質を持たない
  - ⋆ 「目的」レベルでの指示を与えることができない
    - ∗ 手順を詳しく書いた「プログラム」をそのまま実行するだけ
  - ⋆ 予期しない事態に対応できない
    - ∗ 失敗から回復できず、エラーを発生する
  - ⋆ 他者と知識レベルのコミュニケーションをしない
- そのような性質を持つシステムを作るにはどうすれば?
  - ⋆ 人間のやり方を模倣してみるのが有力な手

自律エージェントとは

- 反応型(reactive)のシステム
  - ⋆ 動き続け、周辺環境と相互作用し続ける
    - ∗ 単純に「入力を得て出力を返して終わり」というプログラムはこの性質を満たさない
- 利用者がそのシステムに何らかの仕事を委託し、そのシステムが達成方法を決定する
  - ⋆ 利用者は達成すべき目的だけを与え、詳細に実現方法を指示することはしない
    - ∗ オペレーティングシステム(OS)、銀行のATMシステム、Webサーバ、などはこの性質を満たさない
    - ∗ 囲碁プログラム、ネット上の会話ボット、などもこの性質を満たすとは言い難い

エージェントの特徴

- 環境の中の存在である
  - ★ 環境と相互作用する
    - ∗ 環境から情報を知覚: センサ
    - ∗ 環境への働きかけ(行為): エフェクタ(アクチュエータとも)
    - ∗ 環境は実環境のこともあり、ソフトウェア内の世界のこともある



エージェントの特徴(continued)

- 環境からの情報(知覚)から、いかにして環境への働きかけ(行為)を決めるか
  - プラン(計画)を用いて決める
  - プランは既成の場合も、その場で自分で作る場合もある
- 環境の知覚も、環境への働きかけも不完全(成功するとは限らない)

エージェントの特徴

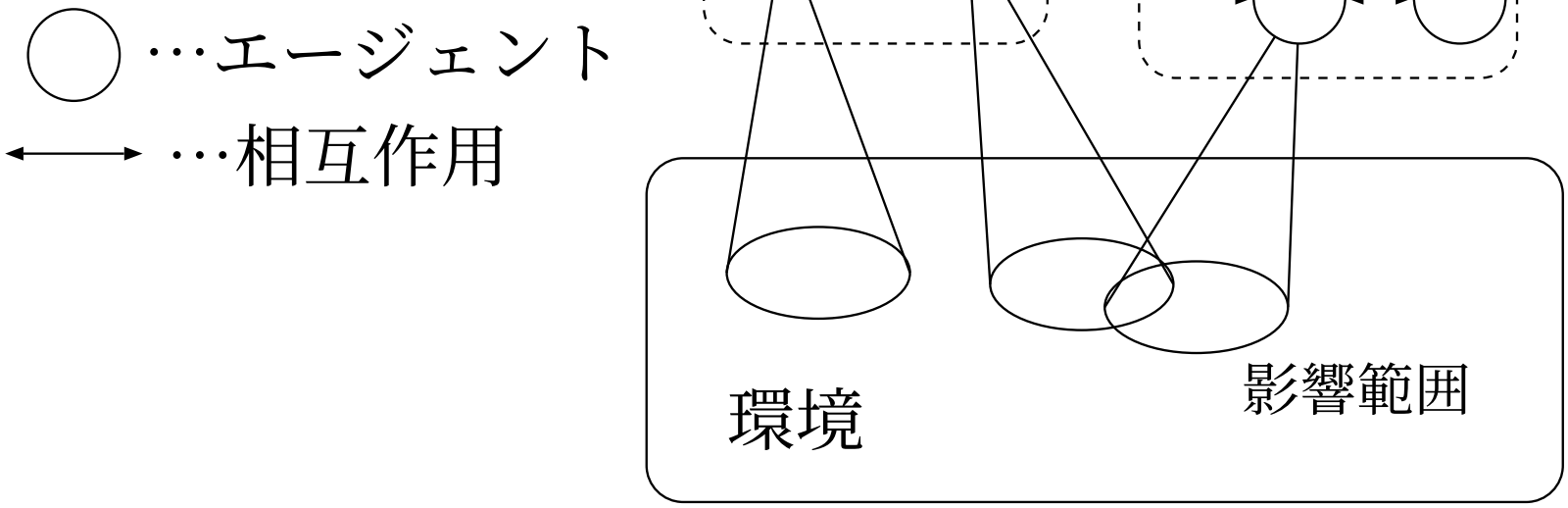
- 自律性
  - ⋆ 完全な「自律システム」は(例えば)人間
  - ⋆ 自ら目的を持ち、目的を達成する方法を選んで行為する
    - ∗ コンピュータシステムの場合「自ら目的を持つ」は難しい場合があるので、「利用者が目的を与え、システムが達成方法を選ぶ」と考える
- 目的志向性
  - ⋆ 「これこれをせよ」(手続き)ではなく「何かを達成せよ」(目的)

エージェントの特徴(continued)

- 反応性
  - ⋆ 環境からの情報に反応して目標や行動を変えたりする
- 社会性
  - ⋆ 他のエージェントと必要に応じて連携・協同する
  - ⋆ 情報(信念、目標、プランなど)を知識としてやりとりする

マルチエージェントシステム

- 単独のエージェントシステムは少ない
- 通常、環境内には他のエージェントが存在する
  - ⋆ 各エージェントにとって、影響を及ぼす範囲がある
  - ⋆ エージェント間で相互作用しあう関係や、組織的な関係もある



エージェントシステムの構築における要請

- 信念、目標、計画(プラン)といった概念を持つこと
- 目的を与えるという形で仕事が委託できること
- 目的指向での問題解決ができること
- 環境に反応して振る舞いを変える機能を持つこと
  - ⋆ 目的指向での問題解決との見通しよい統合も必要
- 知識レベルでのエージェント間のやりとりや協同ができること

特に、エージェントシステムを構築するための手段は、これらの機能を記述できることが求められる

エージェントシステムの構築に向けて

- BDIモデル
  - ★ 自律エージェントのモデル
  - ★ 信念·願望·意図の3つの心的状態を概念として持つ
  - ★ 行為選択の説明に「意図」の概念を使用
    - ＊ 目的達成のための意図を形成して持続し、その意図に基づいて行為を選ぶ
  - ★ 「**意図の理論**」がベース
    - ＊ 人間の目的達成に向けた行為決定機構の模倣

意図の理論

意図の理論

- 哲学者Bratmanが提唱(1987)
- 人間の目的達成に向けての行為選択の説明
- **信念**・**願望**・**意図**の3つの心的状態・・・BDI
- 「**意図**」の概念が鍵
  - ★ 特定の計画のもとに行動しようという心の働き
  - ★ 信念・願望には還元できない
  - ★ **持続性**を持つ(行為の**一貫性**を生む働き)
  - ★ 「未来指向的」意図(この計画でいこう)と「現在指向的」意図(今がその時だ)
- 自律エージェントが備えるべき要件の分析と見ることもできる

意図の(言葉としての)定義

- (1)考えていること。おもわく。つもり。(2)行おうとめざしていること。また、その目的。(広辞苑による)
- 目的を達成するために立案した大きな計画の部分であり、未来または現在、行おうと目指している心的状態
- 意図の理論
  - ⋆ 我々人間は、目的を持って生きている。そして、その目的を達成するために大きな計画を立案する。そして、それら計画から選択して「行おうと目指していること」を意図という心的状態として形成する。

(p. 18, p. 20 は近畿大学「エージェント工学」の授業資料(高田) より、一部改変)

3つの心的状態

信念 エージェントが環境に関して持つ情報。正確・完全とは限らない

- 知覚、あるいは他エージェントとのコミュニケーションなどから得られる
- プランも広義で(目的達成手段に関する)信念の一部

願望 エージェントが到達できたらよいなと思う状況。これだけではエージェントの行為には直結しない

意図 エージェントが目的達成に向かって特定の手段(計画)で取り組もうと決めた状況、ないしはその計画

行為者としての人間

- 我々人間は計画立案する社会的行為者
- 計画立案の問題点
  - ⋆ 信念の不完全性
  - ⋆ 推論能力の有限性
  - ⋆ 個人的および社会的調整の必要性
  - ⋆ 予期せぬ事態が起きうる (詳細な計画は無駄)
- 部分的に計画立案し複雑な目的を達成

実践的推論

- 目的達成に向けた行為を決めるための推論
- 以下の2つからなる
  - ★ 熟考
    - ＊ まず、どのような目的を達成すべきか決定
  - ★ 目的–手段推論
    - ＊ 次いで、その目的をどのような手段で達成するかを決定
- **意図**とは、選んだ手段について「これで達成しよう」と決めた心的状態で、これが行為の原動力となる

実践的推論による行為選択の過程

- 信念と願望をもとに、達成すべき目的を(複数のうちから) 選ぶ(**熟考**)
- その目的を達成するための手段(計画、プラン)を決定(**手段–目的推論**)*

  *計画を複数から選ぶ過程も熟考に含める文献もある

- 手段として選定したプランに対し、「そのプランを実行する」という「**意図**」を形成·保持
- その意図に沿って**行為**
  - ★ その意図は(基本的に)達成まで持続するが、状況に応じては捨てられたり他の意図を選び直したりもする

行為選択の過程(例)

- 喉が渇いた→渇きを癒したい
  - ⋆ 達成すべき目的: 渇きを癒す
  - ⋆ その手段: プラン「ソーダを買って飲む」「紅茶を淹れて飲む」など
    →ここでは「ソーダを買って飲む」を選んだ

- プラン「ソーダを買って飲む」を実行するという意図を形成·保持し、それに沿って行為

(Singh et al., *Formal Methods in DAI: Logic-Based Representation and Reasoning*, 1999. 一部改変)

プラン

プランの構造

- (ラベル ··· プランの名前)
- トリガ·イベント ··· 目標の発生など
- 前提条件 ··· そのプランが選ばれる条件
- **本体** ··· 副目標あるいは基本行為の列
  - ⋆ 副目標 ··· さらに他のプランで再帰的に達成する目標
  - ⋆ 基本行為 ··· それ以上分解できない、直接実行可能な行為

プランの**実行**: 本体を順に達成

- 副目標 ··· その目標を生成し、手段を選んで達成
- 基本行為 ··· 実行

プラン(continued)

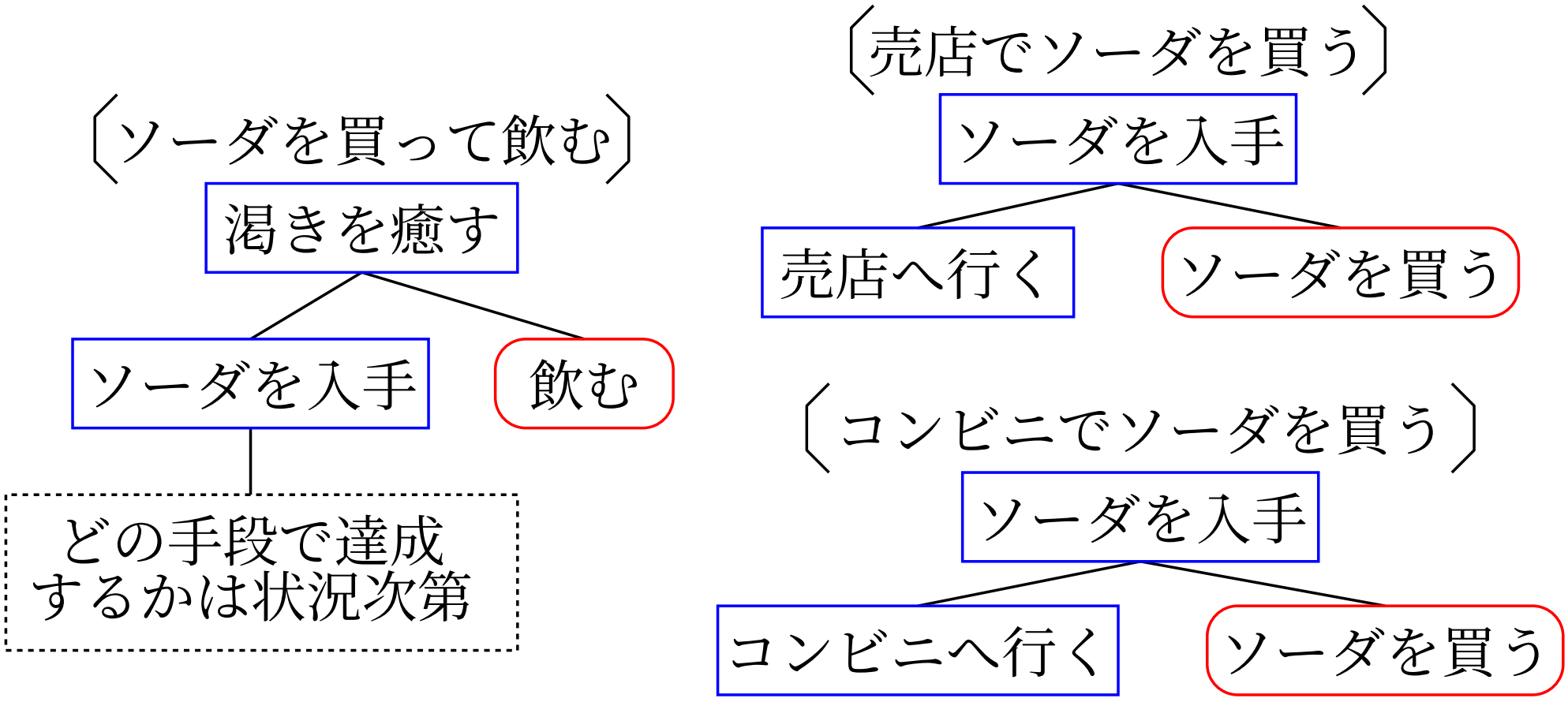
「ソーダを買って飲む」プランの場合

- ラベル: “ソーダを買って飲む”
- トリガ·イベント: 渇きを癒す目標の発生
- 前提条件: ソーダを買うお金がある、など
- **本体**:
  - ⋆ ソーダを入手 ···副目標
  - ⋆ 飲む ···基本行為

その**実行**

- 副目標「ソーダを入手」を達成
- 基本行為「飲む」を実行

プラン(continued)



〈 〉=ラベル　　=(副)目標　　=基本行為

プランの性質

- 部分性
  - ⋆ 最初から目標達成までの全行程を決める必要はない
  - ⋆ 部分的な計画を前もって定め、必要に直面すれば細部を決定する
  - ⋆ ソーダの例
    - ∗ 最初は「ソーダを入手」→「飲む」だけ決定
    - ∗ 「ソーダを入手」する具体的な手段はその時になって決める
  - ⋆ 旅行の例
    - ∗ 最初は「目的地に行く」→「観光する」→「帰る」だけ決定
    - ∗ 観光の具体的な内容は別途決める。あるいは現地でその時に決める

プランの性質(continued)

- 階層性
  - ⋆ 他の副目標を考慮することなく、ある副目標のみ考慮の対象として細部を決定する
- 慣性
  - ⋆ 実行中は再考慮への抵抗が起こる
- 整合性(要請)
  - ⋆ 計画は動的環境でうまく遂行できることが必要
  - ⋆ 信念との整合性があること

未来指向的意図と現在指向的意図

- 達成すべき目標と手段が決まっても、達成すべき時は今でないかもしれない . . . 未来指向的意図
  (例: 帰省するので、12/30は新幹線で広島に行こう)
- 達成すべき時が来れば実行に移す . . . 現在指向的意図
  (例: 12/30が来たから、これから新幹線で広島に行こう)
- 複数の(未来指向的)意図の並立　(マルチタスク)
  - ⋆ 実行すべき時が来れば意識に上る

意図の働き

- 一貫した行為
  - ⋆ 一度手段を決めれば、他の手段は当面考慮しない(再考慮への抵抗·持続性)
    (例: 駅へはこの道で行くと決めたのだから、しばらくはそうしてみよう)
- 失敗からの回復
  - ⋆ 目標が残っていれば、達成のための意図が選び直される
    (例: 駅への道が工事で通れない、駅へ行くには回り道しよう)

意図の再考慮

- 実世界は動的
- 事情が変われば、手段を選び直さねばならないこともある
- **コミットメント戦略** ··· ある意図にどの程度こだわるか
  - ⋆ blind戦略　(達成するまで意図を捨てない)
  - ⋆ single-minded戦略　(達成したか、あるいは達成可能という信念がなくなったとき、意図を捨てる)
  - ⋆ open-minded戦略　(達成したか、あるいは達成するという願望がなくなったとき、意図を捨てる)

合理的な行為選択

心的状態の**整合性**

- 達成できないと信じる意図は形成しない
  - ⋆ 「歯の治療をすれば痛い」と信じるなら「痛くしないで歯の治療をする」という意図は形成しない(Rao et al., *Modeling Rational Agents within a BDI-Architecture*, 1997.)
- 互いに矛盾する意図は形成しない
  - ⋆ 車が1台しかないという信念があるなら、「車で買い物に出かける」と「家族のために車を残して出かける」の両方を意図することはない(Bratman, *Intention, Plans, and Practical Reason*, 1987.)

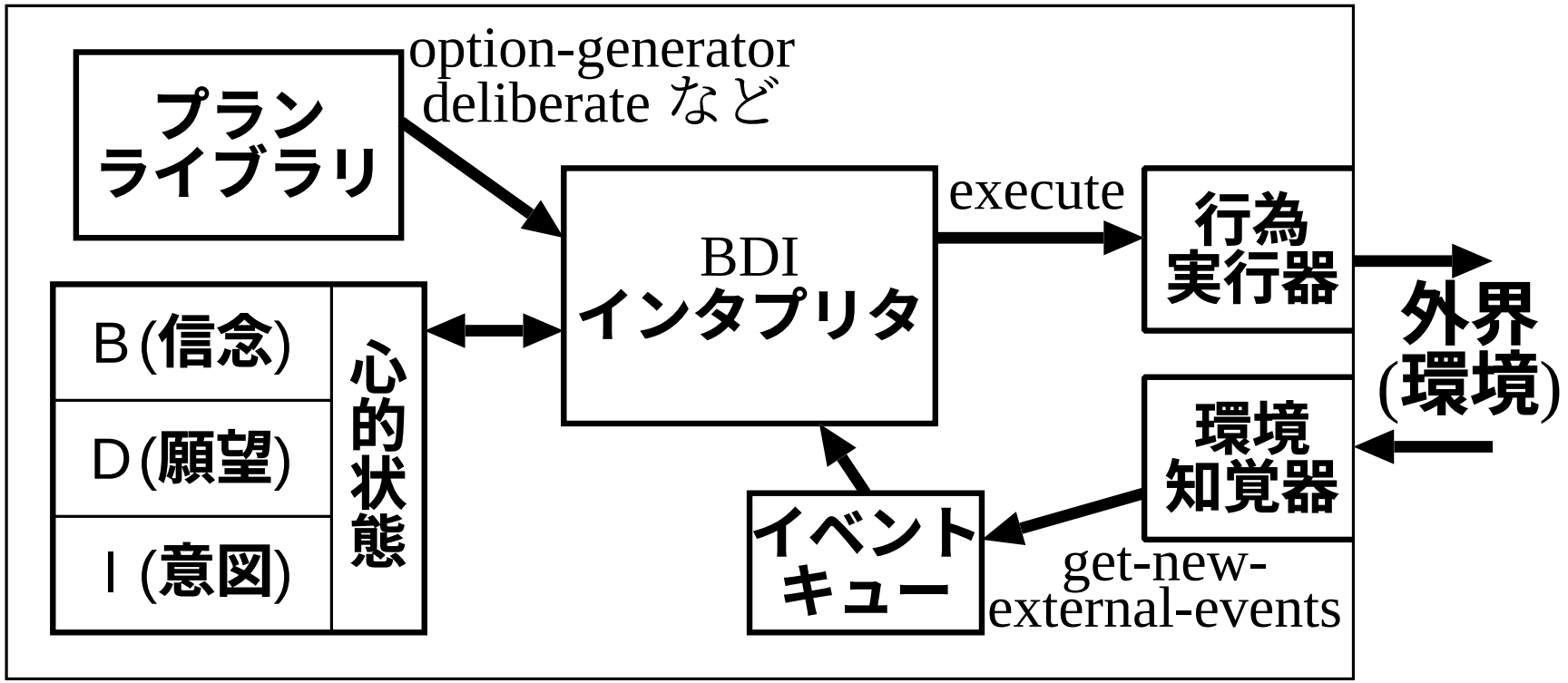
BDIエージェント

BDIモデル

- Rao, Georgeffらが提唱(1991～)
- 意図の理論に基づく、自律·合理的エージェントのモデル
  - ⋆ これに基づくエージェントをBDIエージェントと呼ぶ
- 意図の理論をエージェントの行為決定に適用
  - ⋆ 信念·願望·意図を明示的に持ち、これらを用いて行為決定を行う
- エージェントの振る舞いについて形式的に議論する手段(BDI logic)を持つ

BDIアーキテクチャ

- BDIエージェントの基本アーキテクチャ
- 3つの心的状態、プランライブラリ、センサ・アクチュエータ



⋆ BDIモデルではプランは基本的に既成のものを用いる

BDIインタプリタ

抽象インタプリタ (B:信念 D:願望 I:信念)

BDI-interpreter:
initialize-state();
do
options := option-generator(event-queue, *B*, *D*, *I*);
selected-options := deliberate(options, *B*, *D*, *I*);
update-intentions(selected-options, *B*, *D*, *I*);
execute(*I*);
get-new-external-events();
drop-successful-attitudes(*B*, *D*, *I*);
drop-impossible-attitudes(*B*, *D*, *I*);
until quit.

BDIインタプリタ(continued)

- option-generator
  - ⋆ 願望と信念から、達成すべき目標が発生し、その達成手段の候補を求める(実際には**プランライブラリ**から探す)
- deliberate
  - ⋆ option-generetaorで候補となったものから、実際にどれを選ぶか**熟考**で決定
- update-intentions
  - ⋆ 手段として選定したプランを**意図**とし、現在実行する意図を決定

BDIインタプリタ(continued)

- execute
  - ⋆ 現在実行する意図の(プランの)本体の１ステップを見て、基本行為なら**実行**。副目標ならその目標を発生
- get-new-external-events
  - ⋆ 外部からの**知覚**を受け取る(行為の成否、新たな刺激など)
- drop-{successful,impossible}-attitudes
  - ⋆ 成功した、あるいは不可能とわかった意図の破棄

例題

ソーダの例 [インタプリタ1巡目]

- option-generator ･･･ 渇きを癒したいので、その手段の候補となるプランを列挙
- deliberate ･･･ 列挙した中からプラン「ソーダを買って飲む」に決定
- update-intentions ･･･ それを意図とし、現在実行する意図に選ぶ
- execute ･･･ 意図の(プランの)本体の1ステップ目は副目標「ソーダを入手」なので、それを目標として生成
- get-new-external-events ･･･ 外部の知覚。特に何もなし
- drop-{successful,impossible}-attitudes ･･･ 特に何もせず

例題(continued)

[2巡目]

- option-generator・・・「ソーダを入手」を達成する方法の候補を列挙
- deliberate・・・近くのコンビニで買うプランに決定

⋮

[このあと何巡かするうち、「ソーダを入手」という目標が達成されたとする]

例題(continued)

[n 巡目]

$\vdots$

- update-intentions $\cdots$「ソーダを買って飲む」意図の続行を決定
- execute $\cdots$ 意図の(プランの)本体の次の1ステップは基本行為「飲む」なのでそれを実行
- get-new-external-events $\cdots$ 成功したという知覚を得る
- drop-{successful,impossible}-attitudes $\cdots$「ソーダを買って飲む」意図は成功したので捨てる

実装

- PRS, dMARS (古典)
- AgentSpeak(L) (BDIモデル提唱者Raoによる) → Jason
- Jadex, JACK, etc.
- フリーソフトウェアとして公開されているものも
  - ⋆ 例えばJason(ライセンスはLGPL)やJadex(ライセンスはGPL)がそう

応用

- PRS, dMARS· · · 大規模アプリケーションでの利用
  - ⋆ 航空管理システム · スペースシャトル診断 etc.
- 研究面でも基盤として広く用いられる
  - ⋆ エージェント分野の国際学会AAMAS2014でBDIに触れている発表: 約60件(約600件中)
    - ∗ Antunesら: 願望を自ら獲得 · 発展させるBDIエージェント
    - ∗ Leeら: 社会規範を遵守するBDIエージェント
    - ∗ Dignumら: マルチエージェント向けBDIモデル
    - ∗ Hindriks(招待講演): “· · · most work reported in · · · ProMAS, AOSE, and DALT · · · has taken its inspiration of · · · Belief-Desire-Intention (BDI) agents.”

BDI logic

BDI logic

- **記号論理学**の手法を用いて、BDIエージェントの性質を議論するための論理モデル
- 信念、願望、意図などBDIエージェントに必要な概念を形式的に表現できる

記号論理学

記号による論理学(数理論理学とも)

- 言明を記号で表現
  $\mathsf{father}(\mathsf{namihei}, \mathsf{sazae})\cdots$ 波平はサザエの父
  $\vee\cdots$「または」，$\mathsf{BEL}\,\alpha\cdots$「$\alpha$を信じる」，など
- 自然言語に起因する曖昧さの排除
- 計算機科学との親和性

記号論理学(continued)

言明を記号で表現

- 命題論理

  $\phi \vee \psi \cdots \phi$または$\psi$　$\phi \wedge \psi \cdots \phi$かつ$\psi$

  $\phi \supset \psi \cdots \phi$ならば$\psi$　$\neg\phi \cdots \phi$でない

- 述語論理

  $\forall x\phi \cdots$ 全てのxはϕを満たす

  $\exists x\phi \cdots$ あるxはϕを満たす

- 様相論理(様相命題論理・様相述語論理)

  $\Box\phi \cdots \phi$は必然的に成り立つ

  $\Diamond\phi \cdots \phi$は成り立つ可能性がある　etc.

記号論理学(continued)

自然言語に起因する曖昧さの排除

- 「晴れる」をp、「曇る」をqで表すとき、

  晴れるかまたは曇ることはない

  は$p \vee \neg q$, $\neg(p \vee q)$のどっち?

記号論理学(continued)

計算機科学との親和性

- 「推論」が記号データ上の操作として表現できる
  ...知的システム実現の道具
  - ⋆ 自動演繹
    任意の記号列ϕ, ψに対し、ϕと$\phi \supset \psi$からψを導く
  - ⋆ 論理プログラミング

```
// exp(x, y, z) ... 「xのy乗がz」を表す
exp(x, 0, 1).
exp(x, y, x*w) ← exp(x, y−1, w).
```

このプログラムから2^3が下記の過程を経て求まる

exp(2, 3, 8) ← exp(2, 2, 4) ← exp(2, 1, 2) ← exp(2, 0, 1)

意味論

記号論理学での言明(論理式)の意味(真偽)の定め方

- まず、原子命題($\vee$や$\wedge$などのない、すなわちそれ以上分解できない論理式)の真偽を先に決めておき、

ϕ	ψ	$\neg\phi$	$\phi \wedge \psi$	$\phi \vee \psi$	$\phi \supset \psi$	$\forall x\phi$	$\exists x\phi$
偽	偽	真	偽	偽	真	全てのxについてPが真なら真	あるxについて P が真なら真
偽	真	真	偽	真	真		
真	偽	偽	偽	真	偽		
真	真	偽	真	真	真		

によって他の論理式の真偽を決める

$\phi \supset \psi$は「ϕが真でかつψが偽であることはない」と言い換えられる

例えば$p \supset q$や$p \wedge (p \supset q)$や$p \wedge (p \supset q) \supset q$の意味は、原子命題$p$や$q$の真偽によって以下のようになる

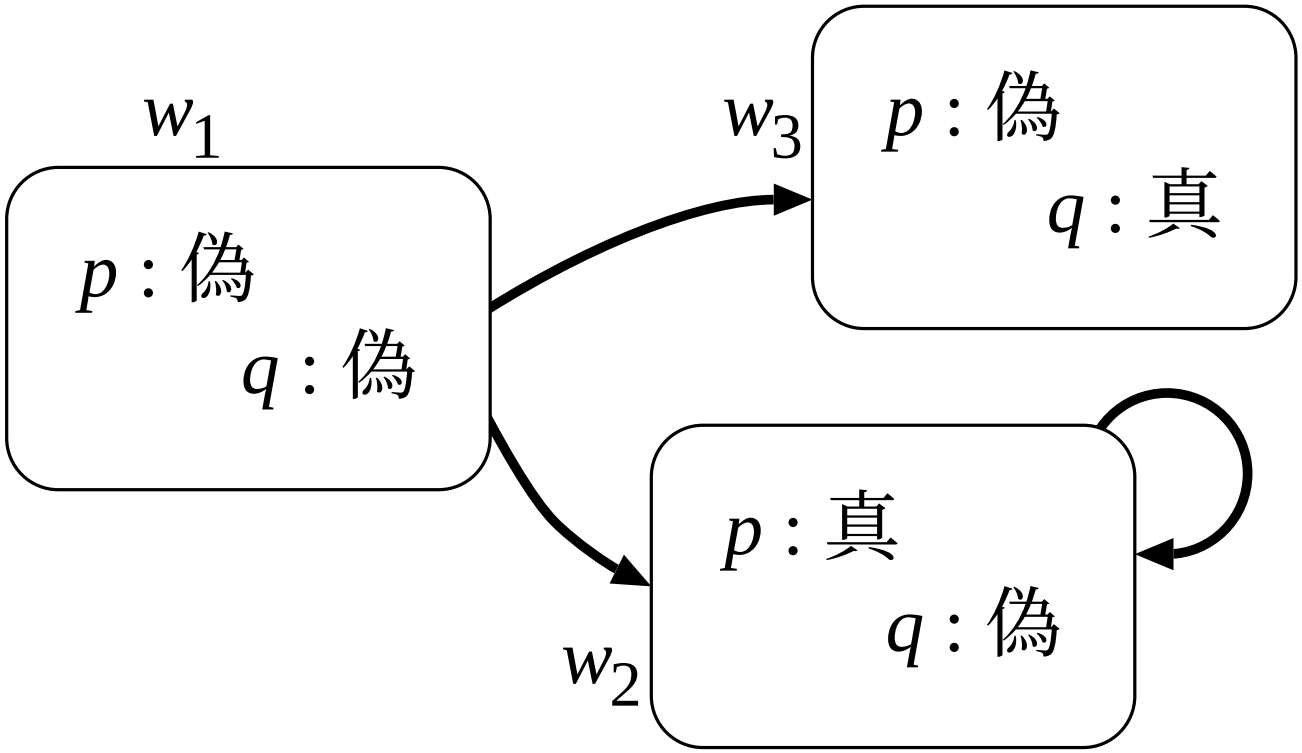
p	q	$p \supset q$	$p \wedge (p \supset q)$	$p \wedge (p \supset q) \supset q$
偽	偽	真	偽	真
偽	真	真	偽	真
真	偽	偽	偽	真
真	真	真	真	真

(ちなみに結合の優先順位は$\neg$, $\wedge$, $\vee$, $\supset$の順)

$p \wedge (p \supset q) \supset q$は、$p$や$q$が真でも偽でも常に真(**恒真**)。他には$p \vee \neg p$などもそう

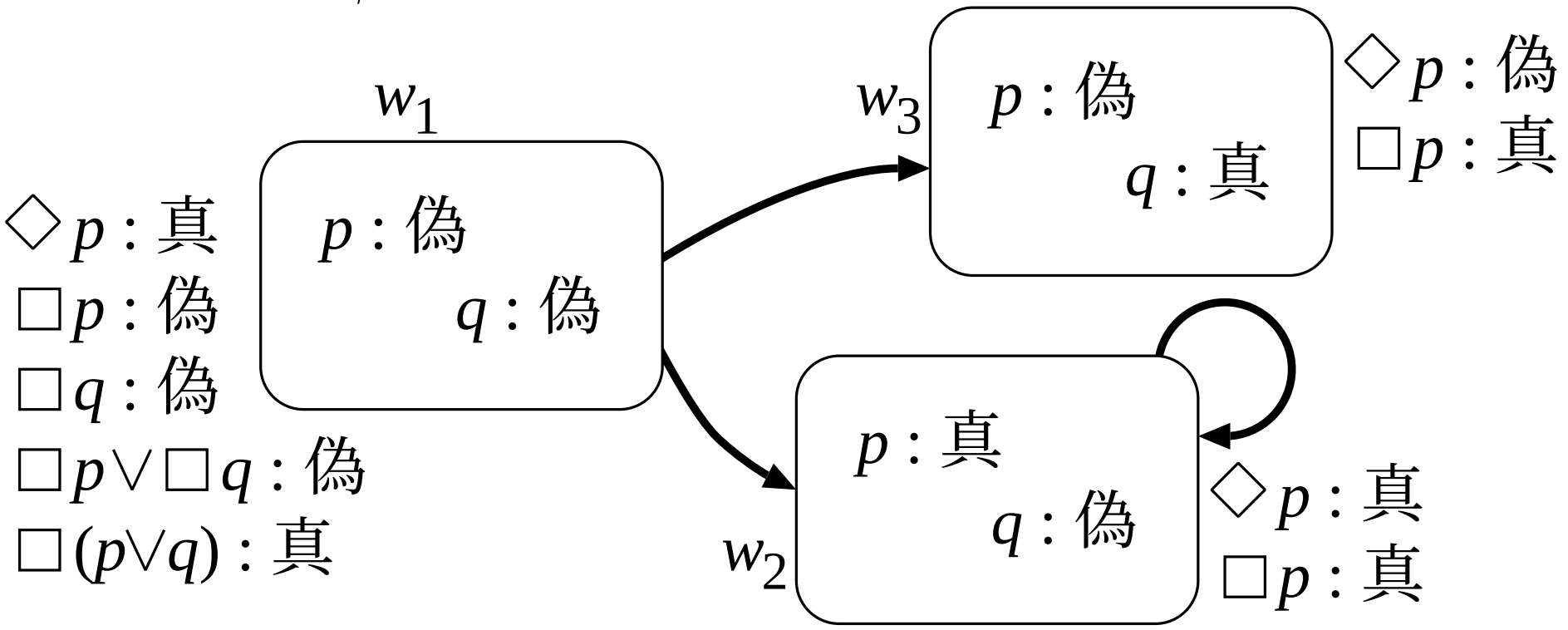
様相論理の意味論

- たくさんの「世界」を考える(**多世界**意味論)
- それぞれの世界で原子命題の真偽を独立に決めておく
- 各世界からどの世界が「見える」かを決める(**可視関係**)



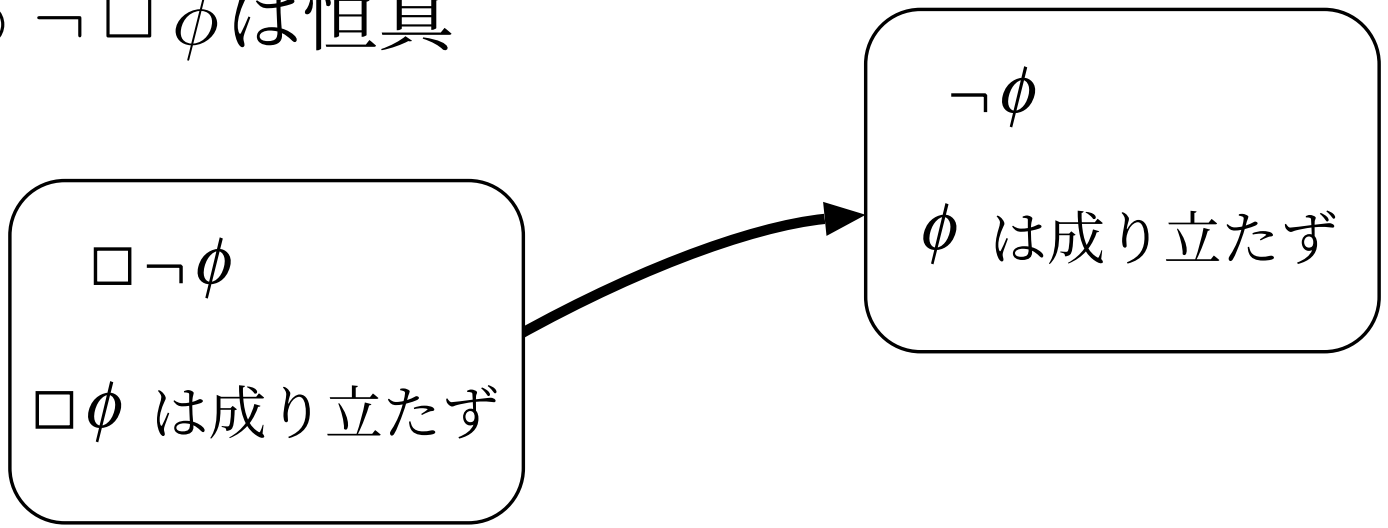
こうした上で以下のように決める

- ある世界wで$\Box\phi$が真であるのは、wから見える**全ての**世界でϕが真であるとき
- ある世界wで$\Diamond\phi$が真であるのは、wから見える**いずれかの**世界でϕが真であるとき



可視関係への制限の入れ方により、□はさまざまな性質を持つ。例えば

1. 可視関係が反射的(自分自身は必ず見える)なら、
   $\Box\phi \supset \phi$は恒真
2. 可視関係がSerial(最低1つの世界が見える)なら、
   $\Box\neg\phi \supset \neg\Box\phi$は恒真



3. 可視関係が推移的なら、$\Box\phi \supset \Box\Box\phi$は恒真

「□」を

- 「信念」と捉えたい場合は2や3
- 「願望」や「意図」と捉えたい場合は2
- 「知識」と捉えたい場合は1, 2, 3
- 「次の時刻」と捉えたい場合は2

を仮定することが多い

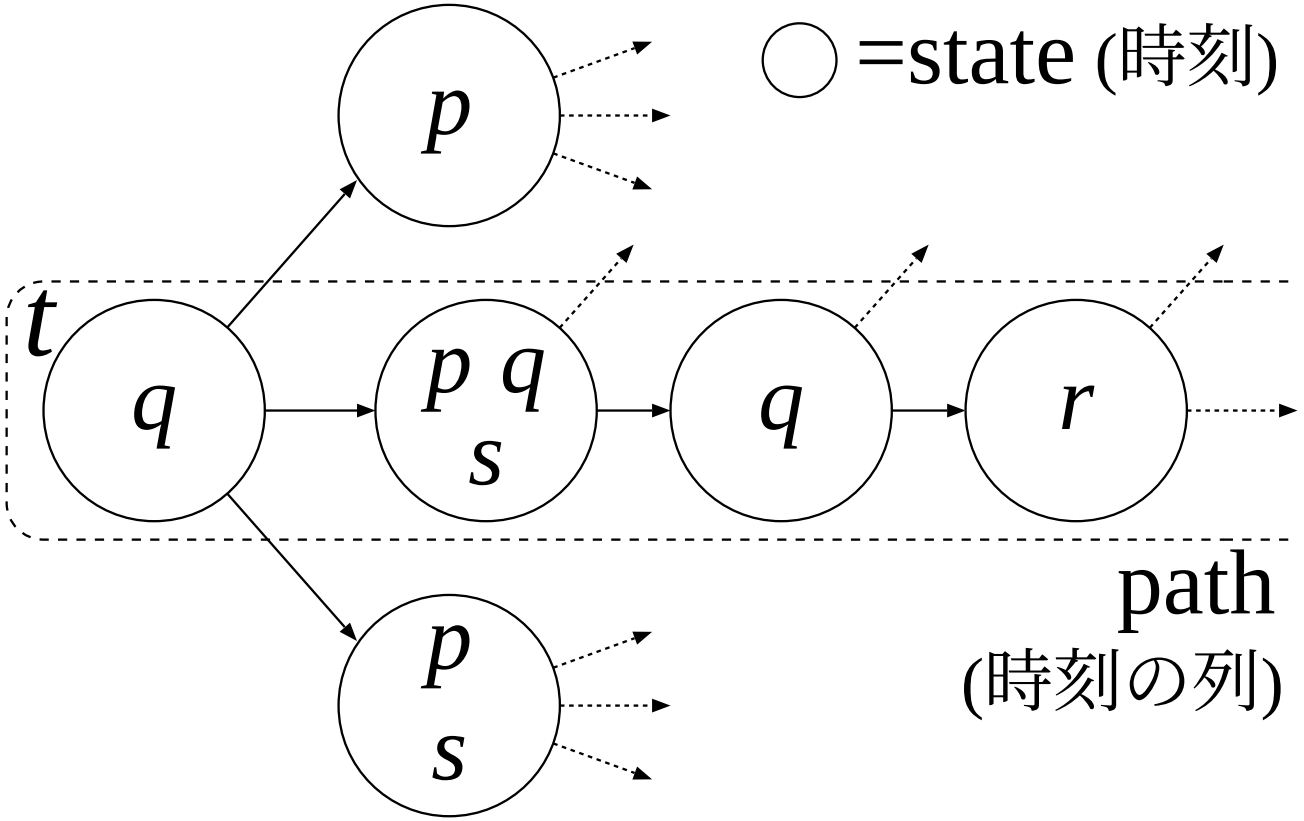
　可視関係を2種類以上導入すると、さまざまな様相を同時に扱える

BDI logic

- BDIモデルの記述に用いる様相論理
- エージェントの性質を形式的に議論
- 以下の様相(心的状態様相・時相)オペレータを持つ
  - ⋆ BELϕ $\cdots$ ϕを信念に持つ
  - ⋆ DESIREϕ $\cdots$ ϕを願望する
  - ⋆ INTENDϕ $\cdots$ ϕを意図する
  - ⋆ Aϕ $\cdots$ 全ての未来でϕ
  - ⋆ Eϕ $\cdots$ ある未来でϕ
  - ⋆ Xϕ $\cdots$ 次の時刻にϕ
  - ⋆ Gϕ $\cdots$ 永遠にϕ
  - ⋆ Fϕ $\cdots$ いつかϕ
  - ⋆ ϕ U ψ $\cdots$ ψが成り立つまでϕ
  - ⋆ $does(e)$ $\cdots$ 基本行為eを実行する

分岐時間様相

BDI logicでは、時間は離散で、未来方向に木構造

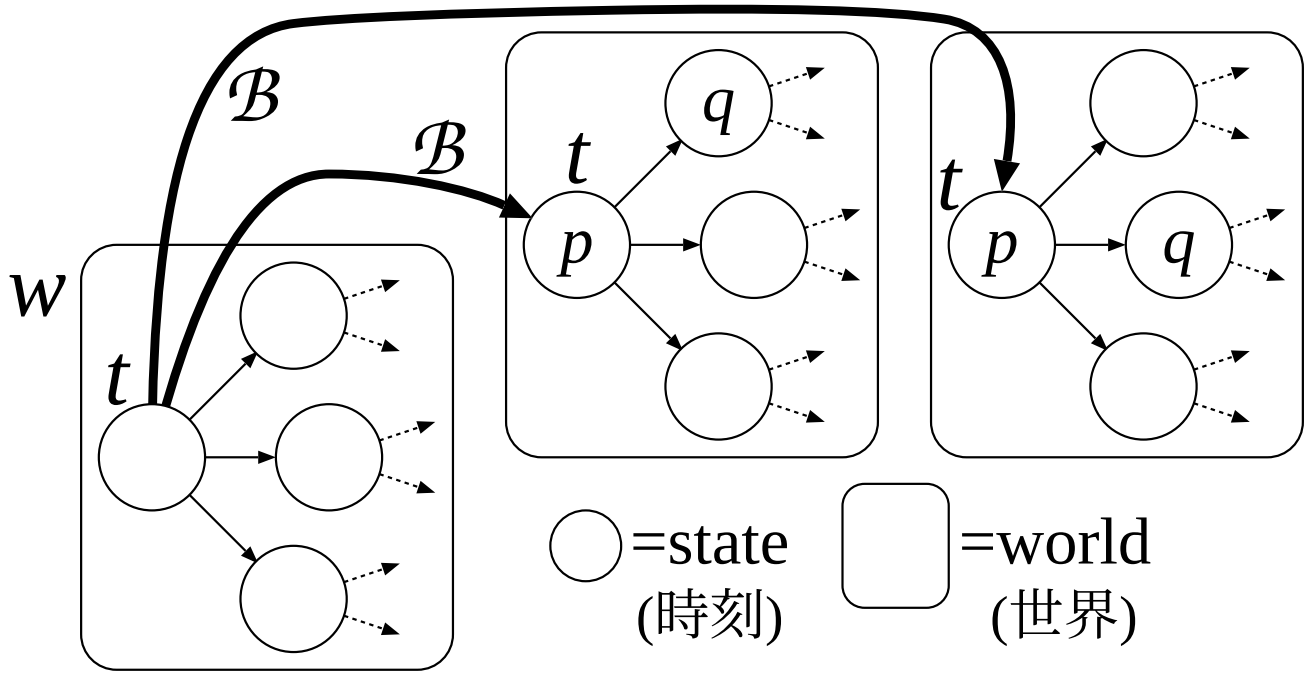


時刻tで真になる論理式の例: $\mathsf{AX}\,p$, $\mathsf{E}(q\,\mathsf{U}\,r)$

心的状態様相(信念・願望・意図)

BDIストラクチャ

- 時間の木が1つの「世界」
- 世界間の可視関係$\mathcal{B}$で表される様相を「信念」、$\mathcal{D}$が同「願望」、$\mathcal{I}$が同「意図」と考える



世界wの時刻tで真となる論理式の例: BELp, BEL EX q

BDI logicによる記述例

INTEND ソーダを買って飲む $\supset$
　INTEND AF ソーダを入手 $\wedge$
　AG(BEL ソーダを入手 $\supset$ INTEND $does$(飲む) $\wedge$
　　AX(BEL $succeeded$(飲む) $\supset$ BEL 渇きが癒される))

INTEND $does$(飲む) $\supset$ $does$(飲む)

- 今「ソーダを買って飲む」意図を持つと、まず「ソーダを入手」する意図を形成する。
その後、「ソーダを入手」したという信念が得られた時、基本行為「飲む」を意図し(従って実行し)、次の時刻に「飲む」が成功したという信念が得られれば、渇きが癒されたと信じる

BDI logicによる記述例(continued)

コミットメント戦略 · · · 意図の持続性

$\mathrm{INTEND}\,\phi \supset \mathrm{A}(\mathrm{INTEND}\,\phi\ \mathrm{U}\ \mathrm{BEL}\,\phi)$

· · · blindコミットメント戦略

$\mathrm{INTEND}\,\phi \supset \mathrm{A}(\mathrm{INTEND}\,\phi\ \mathrm{U}\ (\mathrm{BEL}\,\phi \vee \neg\,\mathrm{BEL}\,\mathrm{EF}\,\phi))$

· · · single-mindedコミットメント戦略

$\mathrm{INTEND}\,\phi \supset \mathrm{A}(\mathrm{INTEND}\,\phi\,\mathrm{U}\,(\mathrm{BEL}\,\phi \vee \neg\,\mathrm{DESIRE}\,\mathrm{EF}\,\phi))$

· · · open-mindedコミットメント戦略

下に行くほど専念の度合が低い

BDI logicによる記述例(continued)

心的状態の整合性

$\mathrm{DESIRE}\,\phi \supset \mathrm{BEL}\,\mathrm{DESIRE}\,\phi$

・・・内省公理 (INTENDやBELに対しても)

$\mathrm{DESIRE}\,\mathrm{EF}\,\phi \supset \mathrm{BEL}\,\mathrm{EF}\,\phi$

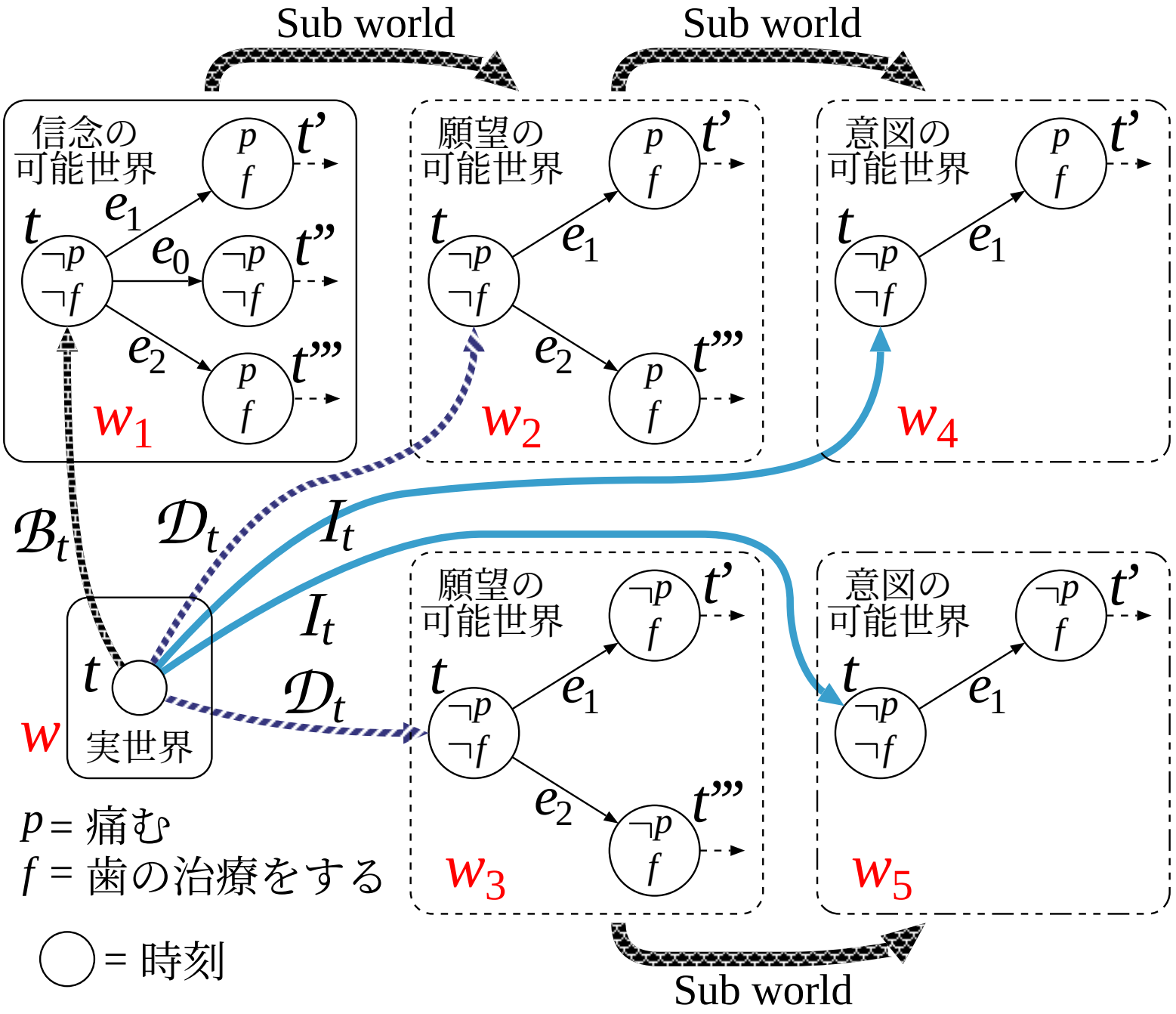
・・・願望と信念の整合性 (INTENDとBELに対しても)

下のものの対偶$\neg\,\mathrm{BEL}\,\mathrm{EF}\,\phi \supset \neg\,\mathrm{DESIRE}\,\mathrm{EF}\,\phi$と、信念の性質$\mathrm{BEL}\,\neg\,\mathrm{EF}\,\phi \supset \neg\,\mathrm{BEL}\,\mathrm{EF}\,\phi$から

$\mathrm{BEL}\,\neg\,\mathrm{EF}\,\phi \supset \neg\,\mathrm{DESIRE}\,\mathrm{EF}\,\phi$

が出る

BDI logicによる意思決定のモデル化



BDI logicによる意思決定のモデル化(continued)

- 信念として持つ時系列中から願望を、さらにその中から意図を選択(Sub world)
- 信念と整合しない願望(痛まずに歯を治したい)や意図は持たない

  $\mathsf{BEL} \neg \mathsf{EF}(f \wedge \neg p) \supset \neg \mathsf{DESIRE}\, \mathsf{EF}(f \wedge \neg p)$
- 願望は信念に引きずられない(痛むことは願望しない)

  $\mathsf{BEL}\, \mathsf{AG}(f \supset p) \wedge \mathsf{DESIRE}\, \mathsf{EF}\, f \supset \mathsf{DESIRE}\, \mathsf{EF}\, p$は成り立たない

エージェントの意思決定過程やその合理性をうまくモデル化できている

BDIモデルの有効性

- (意図の理論で述べられた)人間のものに近い行為決定
  - ⋆ 問題解決に向けた一貫した行為
  - ⋆ プランの部分性
  - ⋆ 複数意図の並行処理
  - ⋆ 合理性、再考慮etc.
- 行為決定のモデル化、(BDI logicによる)形式化

BDIモデルに元々ない部分

- 技能(反射的行為)の獲得と利用(i.e. 機械学習)
- 社会性(社会的義務など)
- プランニング(プランを動的に作成)
- 非合理的な行為(感情、etc.)

BDIモデルは、人間の行為を決めるメカニズムのうち主に「頭で考える」部分を取り上げたもの

BDIモデルに対する拡張

- BDIエージェントに対する拡張の要求
  - ⋆ 例えば
    1. 概念の追加(「能力」「権利」「義務」など)
    2. 異なる行為決定機構
  - ⋆ 2.の例として考えうるもの
    - ∗ 機械学習(例えば強化学習)との結合
    - ∗ プランナの導入

強化学習との結合

- 強化学習 · · · 訓練によって良い行動を学ぶ方法(の1つ)
  - ⋆ BDI: 熟考の部分を担う
  - ⋆ 強化学習: 技能の獲得と利用の部分を担う
- 人間も行為にあたって両者を併用している
  - ⋆ 例: 自転車で駅まで行く際
    - ∗ 最適なルートを経由するプランを熟考で選び、意図として実行
    - ∗ そのルートを巡航するには、倒れないで自転車を漕ぐ技能(学習で獲得)を使う
- 強化学習で獲得したスキルを実践的推論の対象とするBDIエージェント (JAWS2004, 高田)

プランナとの結合

- プランナ・・・プランを生成するソフトウェア
- プランライブラリにないプランの合成
  - ⋆ BDIの熟考ルーチンが必要に応じてプランナを呼び出す
  - ⋆ 例: 強化学習で得た方策をプランナで合成して他の問題に流用 (JAWS2010, 新出)
    - ∗ 人間が、学習で得たスキルを他の問題にも応用することと似ている?

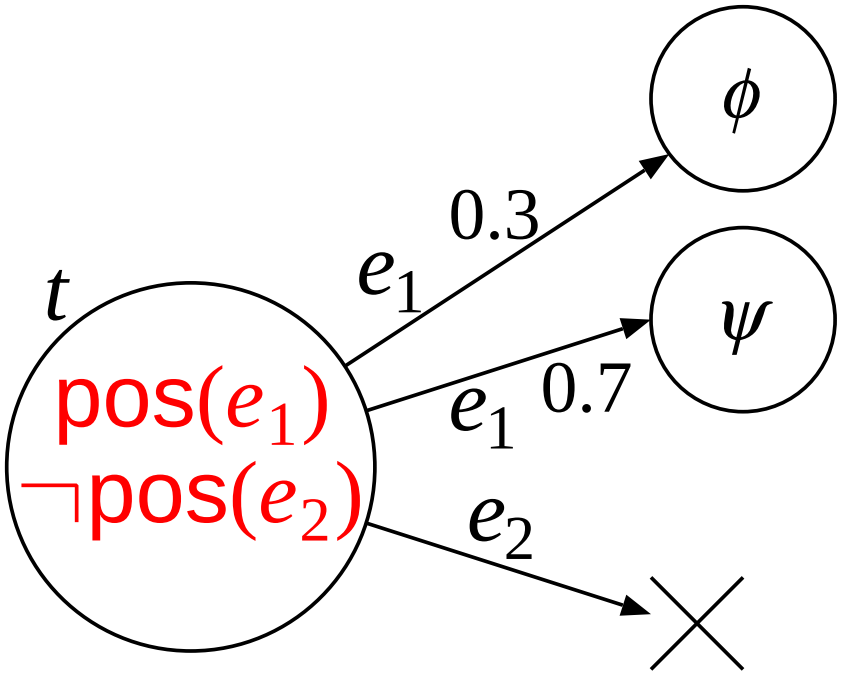
BDIモデルに対する拡張(continued)

- 導入された拡張のモデル化も必要
- ここではBDI logicに
  - ⋆ 強化学習を記述できる拡張
  - ⋆ マルチエージェントへの拡張

  を行う例について紹介する

拡張BDI論理の例 — TOMATOes-P

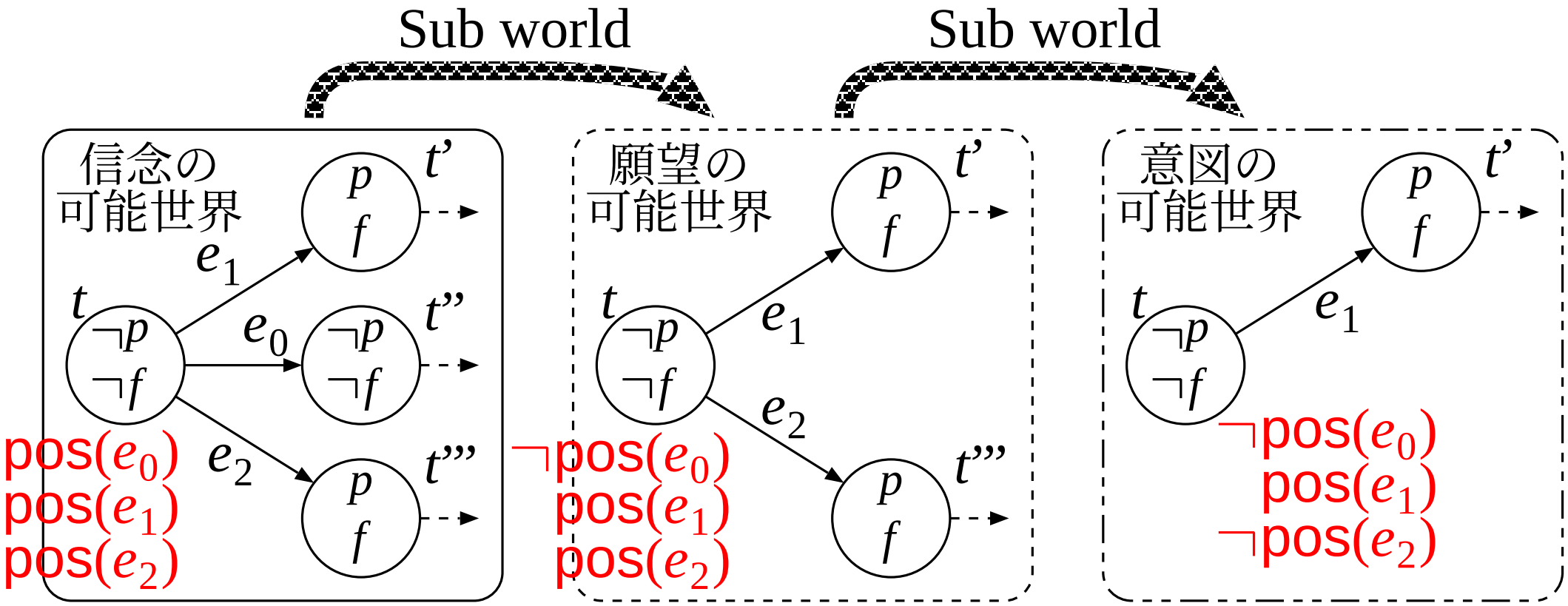
- エージェント毎の心的状態様相オペレータ
- 確率的状態遷移・確率的心的状態オペレータ
- 各時刻でのイベント選択(各イベントの実行の可否)オペレータ



- 不動点オペレータ

イベント選択オペレータの効用

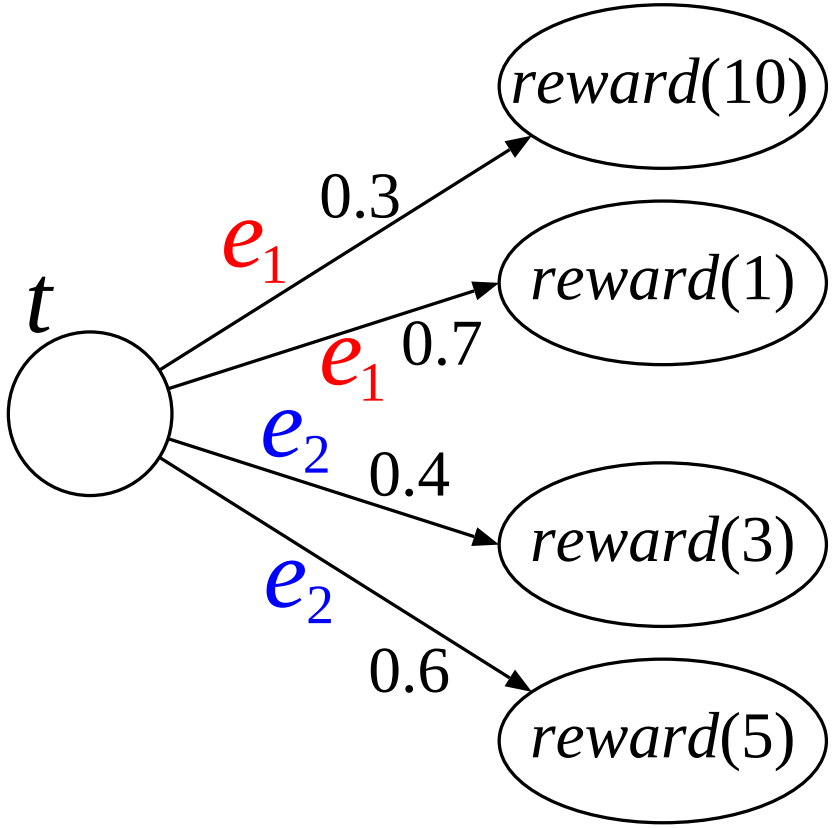
- 意思決定過程をモデル化し直せる



信念・願望・意図の順に可能なイベントを減らしていく

確率的オペレータの効用

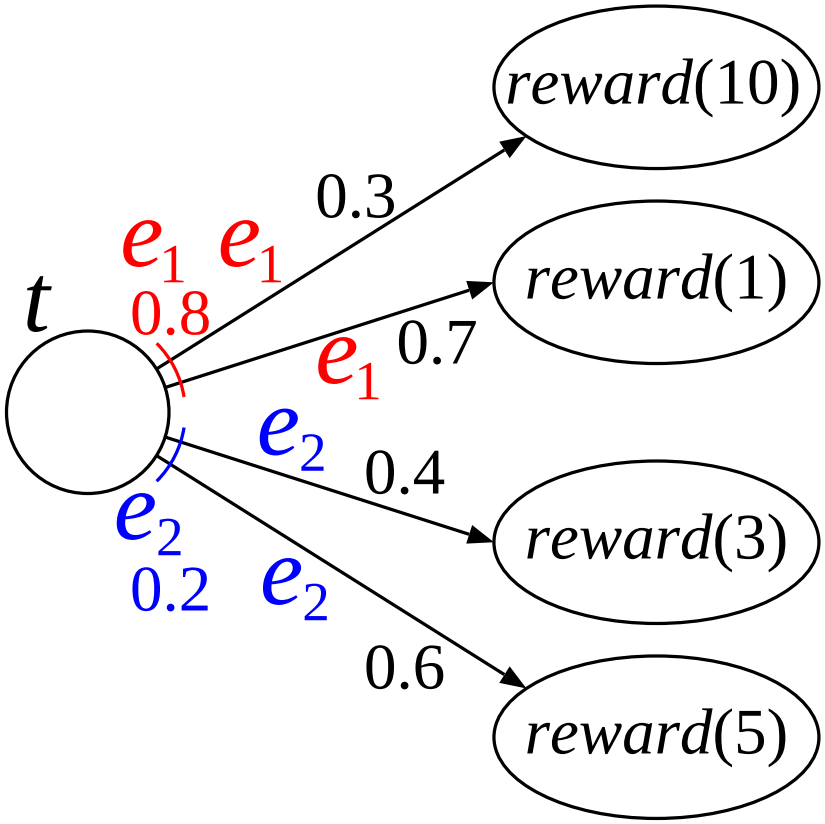
- 強化学習機構との結合を表現



行為e_1とe_2は
どちらが得か?

- 確率的状態遷移の表現能力を要する

- 確率的なイベント選択にも拡張可能



強化学習の行為選択の方策が、意図の選択と同じ「イベント選択」というメカニズムで書ける(どちらも行為決定の一種なのだから自然)

マルチエージェントへの拡張

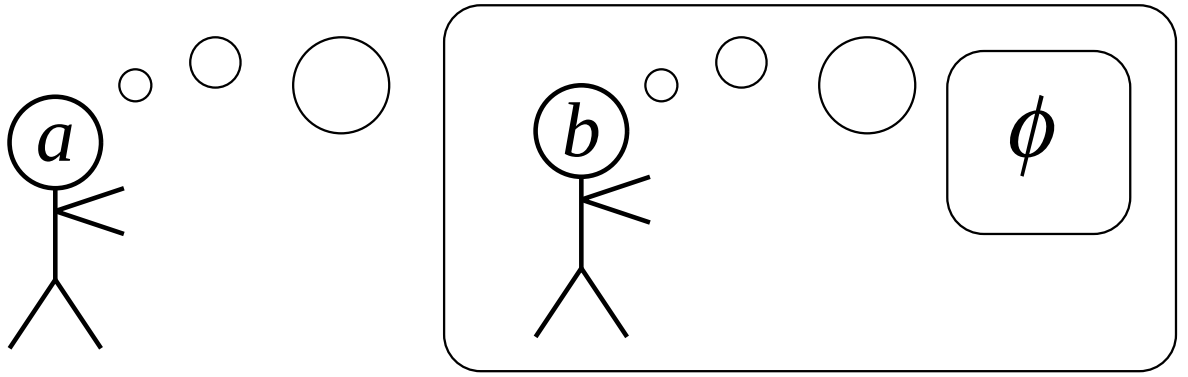
- エージェント毎の心的状態様相オペレータを導入(最初のBDI logicにはなかった)

$$\mathrm{BEL}^a,\ \mathrm{DESIRE}^a,\ \mathrm{INTEND}^a\ (a\text{はエージェント})$$

例:

$\mathrm{BEL}^a\,\phi \cdots$ エージェントaはϕを信じる

$\mathrm{BEL}^a\,\mathrm{BEL}^b\,\phi \cdots$ エージェントaは「エージェントbがϕを信じる」と信じる



不動点オペレータの効用

様々な概念を、それぞれに新たなオペレータを持ち込むことなく、統一的に表現できる

- $X \equiv (\phi \vee \mathsf{EX}\, X)$
  Xと「現在ϕであるか、またはいずれかの未来の1時刻後にXが成り立つ」が等価$\cdots$いずれかの未来でいつかϕが成り立つ(EFϕと同じ)
- $X \equiv (\phi \wedge \mathsf{BEL}^a\, X \wedge \mathsf{BEL}^b\, X)$
  ϕであり、エージェントaとbはそのことを信じ、さらにお互いがそれを信じていることを信じ、さらに$\cdots$(無限の入れ子) $\rightarrow$ 相互信念

- $X \equiv (\phi \vee \mathsf{AX}^{e_1} X \vee \mathsf{AX}^{e_2} X)$

  Xと「現在ϕであるか、またはイベントe_1とイベントe_2のいずれかを実行すると次にXが成り立つ」が等価

  $\cdots$ イベントe_1とe_2のうまい組み合わせで、ϕが成り立つ状態に到達可能

計算機による推論

計算機による推論

- これまでに見てきたBDIエージェントの実現のためには、計算機による推論の能力が必要
  - ⋆ プランの前提条件が現在の信念から導けるか、など

記号論理学での推論

- 推論は、**記号列**(論理式)**に対する操作**として表現(どのような操作を認めるのかは、あらかじめ決めておく—これを「推論規則」という)

例: 推論規則として以下を認めておく(推論規則は、真の論理式からは真の論理式が導かれるように作る)

1. $\forall x\, P$から$P[x := t]$を導く
   ($\forall x$は「全てのxについて」、$P[x := t]$はPの中のxにtを代入したものを表す)
2. Pと$P \supset Q$からQを導く

記号論理学での推論(continued)

すると次のような推論ができる(横線の上の論理式から下の論理式を導く)

$$\dfrac{\mathrm{fine}(\mathrm{today}) \quad \dfrac{\forall x(\mathrm{fine}(x) \supset \mathrm{dry}(x))}{\mathrm{fine}(\mathrm{today}) \supset \mathrm{dry}(\mathrm{today})}}{\mathrm{dry}(\mathrm{today})}$$

記号論理学とコンピュータ

- 記号列の操作はコンピュータで行える
  - ⋆ 定理自動証明(1960年頃～): 一定の条件のもとで成果
  - ⋆ 記号論理学とコンピュータ·情報科学との結びつき
- 推論をコンピュータにさせられるなら、コンピュータによる「思考能力」が実現できるのでは?
- しかし、人の思考をコンピュータに真似させることはそう簡単ではないことが次第にわかった(曖昧な判断、常識による判断、推論の筋道の探索···)
- 部分的な知能の実現ならコンピュータにもできるので、それを(少しずつでも)いかに人に近づけていくかが課題

論理プログラミング

- 推論をコンピュータに行わせる過程は、**データ処理**や**計算**の道具としても使えることがわかった
  - ⋆ (例)「0の階乗は1である」「$x-1$の階乗がzならば、xの階乗はzのx倍である」→「3の階乗は6である」

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{fact}(0,1)}\quad \dfrac{\forall x(\mathrm{fact}(x-1,z)\rightarrow \mathrm{fact}(x,z\cdot x))}{\mathrm{fact}(0,1)\rightarrow \mathrm{fact}(1,1)}}{\mathrm{fact}(1,1)}\quad \dfrac{\forall x(\mathrm{fact}(x-1,z)\rightarrow \mathrm{fact}(x,z\cdot x))}{\mathrm{fact}(1,1)\rightarrow \mathrm{fact}(2,2)}}{\mathrm{fact}(2,2)}\quad \dfrac{\forall x(\mathrm{fact}(x-1,z)\rightarrow \mathrm{fact}(x,z\cdot x))}{\mathrm{fact}(2,2)\rightarrow \mathrm{fact}(3,6)}}{\mathrm{fact}(3,6)}$$

Prolog言語

- 論理プログラミングの考え方によって最初に作られたプログラミング(1972年)
- 今も論理プログラミング言語の代表的存在
- 特徴
  - ★ 記号処理言語(「**記号**」を1つのデータとして扱える)
  - ★ 宣言的プログラミング(プログラムは「**知識**」(××ならば○○である)の集まり)
  - ★ プログラムの実行は推論規則による「**推論**」

Prolog言語: 例

プログラム例

```
fine(today).
dry(X) :- fine(X).
```

- 「fine(today)」は「todayという**物**がfineという**性質**を持つ」を表す。このような1つの知識を「**節**」と呼ぶ。

  節の末尾には必ず「.」が付く。

- A :- B.は「BならばAである」を表す(「Aを示すには、Bを示せればよい」と捉えてもよい)。これも1つの知識なので「**節**」。

  もしA :- $B_1, B_2, \cdots, B_n$.の形なら「B_1かつB_2かつ$\cdots B_n$ならばAである」の意。論理式で表記すると$B_1 \wedge B_2 \wedge \cdots \wedge B_n \rightarrow A$(に必要なだけ$\forall$がついたもの)に相当。上の例だと$\forall X(\text{fine}(X) \rightarrow \text{dry}(X))$に相当

Prolog言語: 例

- 「:-」のない節を「**事実**」、ある節を「**規則**」と呼ぶ

  注: 「規則」とは「ならば」のある論理式に相当する節のことであり、推論規則とは異なる。

- 括弧の前のfineやdryは性質を表す語であり、これらを「**述語**」と呼ぶ。括弧の中のtodayやXは物を表す語であり、これらを「**項**」と呼ぶ

  「X」のように、大文字で始まる単語は「**変数**」として扱われる。変数でないものは大文字で始めてはいけない。

Prolog言語: 例

実行例

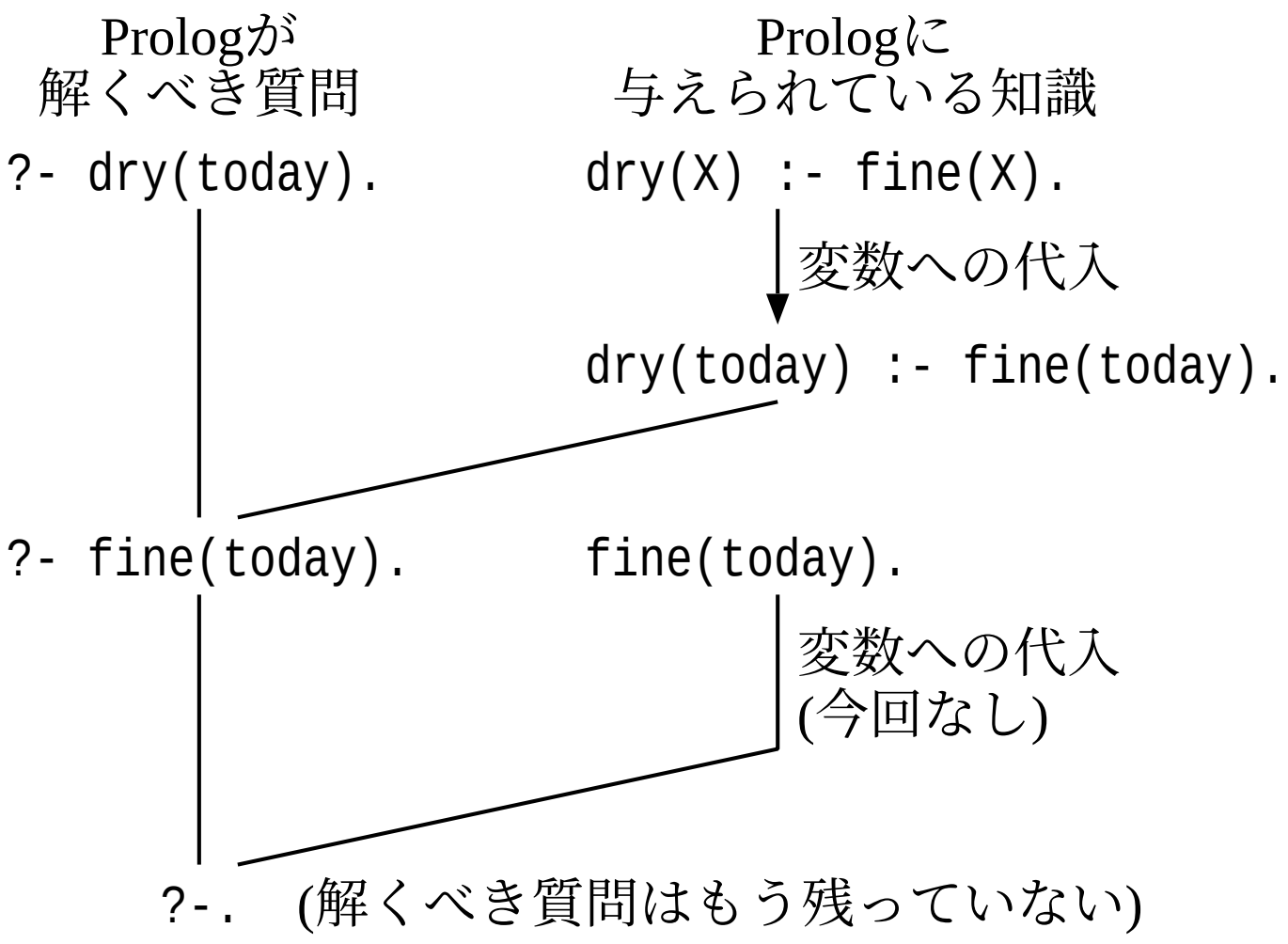
```
?- dry(today).
true.
```

- 「?-」の後ろに来るものは、ユーザからPrologへの**質問**を表す。
- Prologは質問にtrue(真)と答えている。これがProlog プログラムの実行。

  Prologシステムの種類によっては「true」でなく「yes」と答えるものもある

Prolog言語: 例

Prologがユーザの質問に答えるまでの過程



解くべき質問を、持っている知識を使って、別の質問に置き換える。この繰り返し

得たい結論から、その結論を出せる前提へと、逆向きに推論を行うので、この方法を「後ろ向き推論」と呼ぶ

後ろ向き推論のやり方

解くべき質問に対して

- その質問に直接答えられる「事実」を知識として持っているなら、それで終了
- その質問に答えられる可能性のある「規則」を知識として持っているなら、その規則の「:-」の右側を新たな質問として解こうとしてみる

  そのような知識を見つけるには、答えたい質問と同じもの(または変数への代入によって同じにできるもの)を「:-」の左側に持っている規則を探せばよい。

(余談)

「後ろ向き推論」は人間もしばしば使っている

- 「Aという問題を解きたい」「Aを解くにはBが解ければよい」→「じゃあBが解けないかな」という考え方は人間でもよくやる
- しかし、人間の場合はその逆(前向き推論＝既に知っている手がかりから新たなヒントを見つけ出そうとする)も程よくミックスしながら使っているが、Prologは後ろ向き推論だけ

もう少しだけ本格的な例

サザエさんの家族関係のプログラム

```
female(fune).
female(sazae).

male(namihei).
male(masuo).
male(katsuo).
male(tara).

mother(fune, sazae).
mother(fune, katsuo).
mother(sazae, tara).

father(namihei, sazae).
father(namihei, katsuo).
father(masuo, tara).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
son(X,Y) :- parent(Y,X), male(X).
```

プログラムは以下の知識からなる

- 「サザエは女性である」「波平はサザエの父である」などの事実
- 「親」という関係に関する2つの知識「XがYの父ならば、XはYの親である」「XがYの母ならば、XはYの親である」
- 「息子」という関係に関する知識「YがXの親で、かつXが男ならば、XはYの息子である」

プログラムを読み込んで質問をしてみると

- 波平はサザエの父ですか

```
?- parent(namihei, sazae).
true.
```

- 波平はサザエの親ですか
- 波平はマスオの親ですか (`false`になる)
- カツオは波平の息子ですか

変数入りの質問

```
?- parent(X, katsuo).
X = fune ;
X = namihei ;
false.
```

「誰がカツオの親ですか」という意味の質問になる。「;」を押すと次の解が出力され、他に解がなくなると「false.」が出る。

新たな関係の定義

```
/* XはYの兄弟姉妹である */
sibling(X,Y) :-
        parent(Z,X), parent(Z,Y), \=(X,Y).

/* XはYの祖先である */
ancestor(X,X).
ancestor(X,Y) :-
        parent(Z,Y), ancestor(X,Z).
```

組み込み述語の利用

Prologにはいくつかの「組み込み述語」(人間が定義を読み込ませなくても、あらかじめPrologが定義を知っている述語)が用意されている。一例として、「>」や「is」などがある。先ほど出てきた「\=」も実は組み込み述語。

```
?- >(5, 3).
true.
?- >(5, 7).
false.
```

```
?- is(7, 3+4).
true.
?- is(X, 3+4).
X = 7.
```

「>」は数の大小を判定する組み込み述語、「is」は第2引数の数式の計算結果が第1引数の数と等しくなるかどうかを判定する組み込み述語である。

数の計算(continued)

isを使うと、数の計算ができる。次は階乗の計算の例。

```
fact(0, 1).
fact(X, Y) :-
  >(X, 0), is(Xminus1, X-1), fact(Xminus1, Z), is(Y, Z*X).
```

あるいは

```
fact(0, 1).
fact(X, Y) :-
  X > 0, Xminus1 is X-1, fact(Xminus1, Z), Y is Z*X.
```

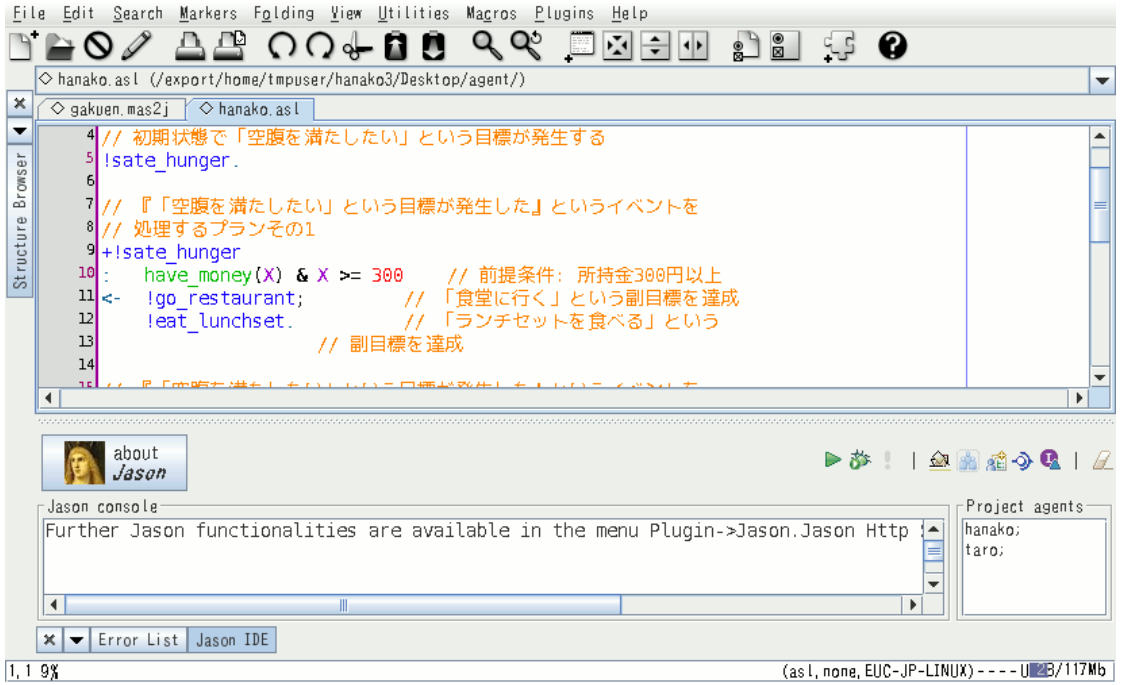
「>(X,0)」では直感的にわかりにくいので、「X > 0」と書いてもいいことになっている。isについても同じ。

実行例

```
?- fact(4, X).
X = 24 ;
false.
```

Jason

- BDIアーキテクチャに基づくエージェント開発の道具として、(BDIアーキテクチャ提唱者の)Raoらが作ったソフトウェア



- エージェントのプランや信念などを記述しておき、エージェントを動かす
- プランの前提条件が満たされているかなどの推論には、Prologと同様の機構が使われている

Jasonでのエージェント記述

- エージェントのプランや、初期状態での信念・目標などを記述
- 1つのプランは原則として「トリガイベント：前提条件 <- 本体」の形(「: 前提条件」は省略する場合もあり、その場合は無条件)
- プランの本体の中身は

  基本行為または副目標;

  ...

  基本行為または副目標.

  の形
- 「+!」で始まるものは、目標の発生を表す「イベント」
- 「!」で始まるものは、目標(または副目標)の発生

Jasonでのエージェント記述(continued)

エージェントhanako

```
// 初期状態で「400円持っている」という信念を持っている
have_money(400).

// 初期状態で「空腹を満たしたい」という目標が発生する
!sate_hunger.

// 『「空腹を満たしたい」という目標が発生した』という
// イベントを処理するプランその1
+!sate_hunger
:   have_money(X) & X >= 300  // 前提条件: 所持金300円以上
<-  !go_restaurant;       // 「食堂に行く」という副目標を達成
    !eat_lunchset.        // 「ランチセットを食べる」という
                          // 副目標を達成
```

```
// 『「空腹を満たしたい」という目標が発生した』という
// イベントを処理するプランその2
+!sate_hunger
:   have_money(X) & X < 300
<-  !go_restaurant;
    !eat_sandwich.

// 『「食堂に行く」という目標が発生した』という
// イベントを処理するプラン
+!go_restaurant    // 前提条件が省略されると「無条件」扱い
<-  .print("going to the restaurant").
    // ここでは実際は画面に何がしか出力しているだけ
    // .printは出力を行う基本行為

// 『「ランチセットを食べる」という目標が発生した』という
// イベントを処理するプラン
+!eat_lunchset
<-  .print("eating lunch set").

+!eat_sandwich  <-  .print("eating sandwich").
```

Jasonでのエージェント記述(continued)

エージェントtaro

```
have_money(300).
!sate_hunger.

+!sate_hunger
:   have_money(X) & X >= 200
<-  !go_restaurant;  !eat_curry;  !eat_noodle.

+!sate_hunger
:   have_money(X) & X < 200
<-  !go_restaurant;  !eat_noodle.

+!go_restaurant <- .print("going to the restaurant").
+!eat_curry     <- .print("eating curry").
+!eat_noodle    <- .print("eating noodle").
```

Jasonでのプランの選択

- 目標が発生したら、その目標が「発生した」という**イベント**が起きる
- そのイベントと一致する**トリガイベント**を持つプランが、意図として選ばれる候補となる
- 候補の中から、**前提条件**が現在の状況下で真であるもの1つが「意図」として選ばれる
  - 2つ以上ある場合は、標準的にはそのうち最初のものが選ばれるが、変更も可能

以上は、現在の状況から、プランに記述された規則によって行為を決めている。このような「行為を決めるための推論」が「**実践的推論**」であり、事実のみに関する推論と区別される。

Jasonでのプランの選択と実行の例

エージェントhanakoの例だと

- sate_hungerという目標が発生(!sate_hanger)
- それによって、+!sate_hungerというイベントが発生
- そのイベントをトリガイベントに持つプランを探す
- そのようなプランの中で、前提条件が満たされるものが選ばれ「意図」となる
- 意図はそれ1つしかないので、本体を実行する
- 本体の実行は、まず副目標「go_restaurant」を発生させて達成
- 次に「eat_lunchset」という副目標を発生させて達成

目標を副目標に置き換えていく過程は、「ある目標の達成を、プランを使って、別な目標の達成に帰着する」というものであり、Prologでの後ろ向き推論の過程(「ある質問に答えることを、規則を使って、別なある質問に答えることに帰着する」)に似ている

Jasonを使ったエージェント開発やロボット開発の実例

会場にて動画で実演(研究室で学生が作ったものです)

問題点として、マスからマスへの動きが不正確で、しばしば手動でマスの中央に戻してやらねばならない点が見てとれる。

実世界で自律的に動くロボットの実現には、精密な動作の実現(ロボティクス＝機械としてのロボットの制御技術)と、考える力の実現(計算機による思考)のどちらもが重要であることがわかる。

BDIさえあれば人間らしい行動が実現できるか

決してそんなことはない

- BDIが提供するのは行動選択の枠組みだけ。プランを選んだり、状況の変化に応じて選び直したりする部分は結局自分で作らないといけない
- 人間の行動決定は、目標を決めてその達成方法を考えて…という「熟考」だけではない。反射的な行動(例えば、自転車がこけかけたら逆にハンドルを切る)などを機械学習で獲得することなども必要

通しのまとめ

- 人間のような「目的達成のために行為する」能力をコンピュータで実現するには、意図の理論に基づいて実践的推論を実現する、BDIモデルが有用である
- 記号論理学とコンピュータの結びつきは、「考えて行動する力」のコンピュータによる実現に、大きな力となる

考えてみたいこと

- 普段の生活で我々が行っている実践的推論について考えてみよう。それをコンピュータでの自律エージェントにどの程度模倣させられるだろうか。
- BDIモデルによる自律エージェントは、人間が行っている実践的推論を、どの程度近似できているだろうか。それは、人間と同じになれるだろうか。そうなるにはどこが困難だろうか。
- 鉄腕アトムやドラえもんは、自律エージェントと言えるが、それだけではない。将来それらは実現するだろうか。今現在、どの程度実現できているだろうか。

参考文献

- 山川他: 特集: 意図研究のスペクトル, 人工知能学会誌, Vol. 20, No. 4, pp. 357–455, 2005. 「意図」に関する解説記事、BDIモデルの解説もあり
- Bratman: *Intention, Plans, and Practical Reason*. Harvard University Press, 1987. (門脇他訳, 意図と行為—合理性、計画、実践的推論—, 産業図書, 1994.) 「意図の理論」の原典
- Rao et al.: Modeling Rational Agents within a BDI-Architecture. In *Reading in Agents*, Morgan Kaufmann, pp. 317–328, 1997. BDIモデルによる合理的エージェントの形式化
- Singh et al.: Formal Methods in DAI: Logic-Based Representation and Reasoning. In *Multiagent Systems*, The MIT Press, pp. 331–376, 1999. BDIアーキテクチャの解説

- Bordini et al.: *Programming multi-agent systems in AgentSpeak using Jason*, Wiley, 2007. BDIエージェントとJasonについて
- Sutton et al.: *Reinforcement Learning: An Introduction*, The MIT Press, 1998. (三上他訳, 強化学習, 森北出版, 2002.) 強化学習の教科書
- Ghallab er al.: *Automated Planning: Theory & Practice*, Morgan Kaufmann, 2000. プランニングの教科書
- その他、本文中で言及した論文・授業資料類

付録1: 強化学習

強化学習

- 機械学習の一手法
- 自分の行動に対する環境からの結果を見て、有利な行為の選択を学ぶ
  - ★ 「状態」と「行動」の対に対する推定価値を求める

n本腕スロットマシン問題

- レバーを3本持つスロットマシンがある
- レバーを1回押すたびに以下の報酬が得られる
  (エージェントは報酬の決まり方を知らない)

レバー1	レバー2	レバー3
確率0.7で報酬1	確率0.6で報酬2	確率0.2で報酬4
確率0.3で報酬0	確率0.4で報酬-1	確率0.3で報酬3
		確率0.5で報酬-2
平均報酬0.7	平均報酬0.8	平均報酬0.7

- エージェントはどのようにレバーを押すべきか?

n本腕スロットマシン問題(continued)

- それぞれのレバーの平均報酬を推定し、平均報酬が一番高いレバーを押しつづければよい
- しかし、レバーの平均報酬を推定するには、どのレバーも何回か押してみなければならない
- そこで
  - ⋆ 一定の確率(例えば0.1)でランダムにレバーを押す
  - ⋆ その他の場合には、これまでに平均的な報酬の推定が最も良かったレバーを押す
  - ⋆ 報酬の推定値は、そのレバーを実際に試して得た報酬の平均とする
- これをε-greedy法という (他にもsoftmax法などが知られる)

強化学習とは

- 「どの選択肢を選ぶのが最善か」を教えてくれる存在はいない(教師なし学習)が
- 実際に試すことで何らかの報酬が得られる

  …という場合に
- 「探索」と「(推定値の)利用」をバランスよく行うことでうまい行動をとろうとするやり方
- 「何度もトライして、よい結果を得るやり方を学んでいく」点で生物の行う学習行動とも似ている

再びn本腕スロットマシン問題

- 「300回ほどランダムに試して各レバーの平均報酬を推定し、以後は報酬が一番良かったものをずっと押し続ける」のはよい方法か?
- これでは、途中から各レバーの報酬の決まり方が変わる(動的環境)場合に対応不可
- ε-greedy法はそういう場合にも対応できる
  - ⋆ 途中で報酬が変わると、平均報酬の推定は徐々に新しい報酬の決まり方を反映していく

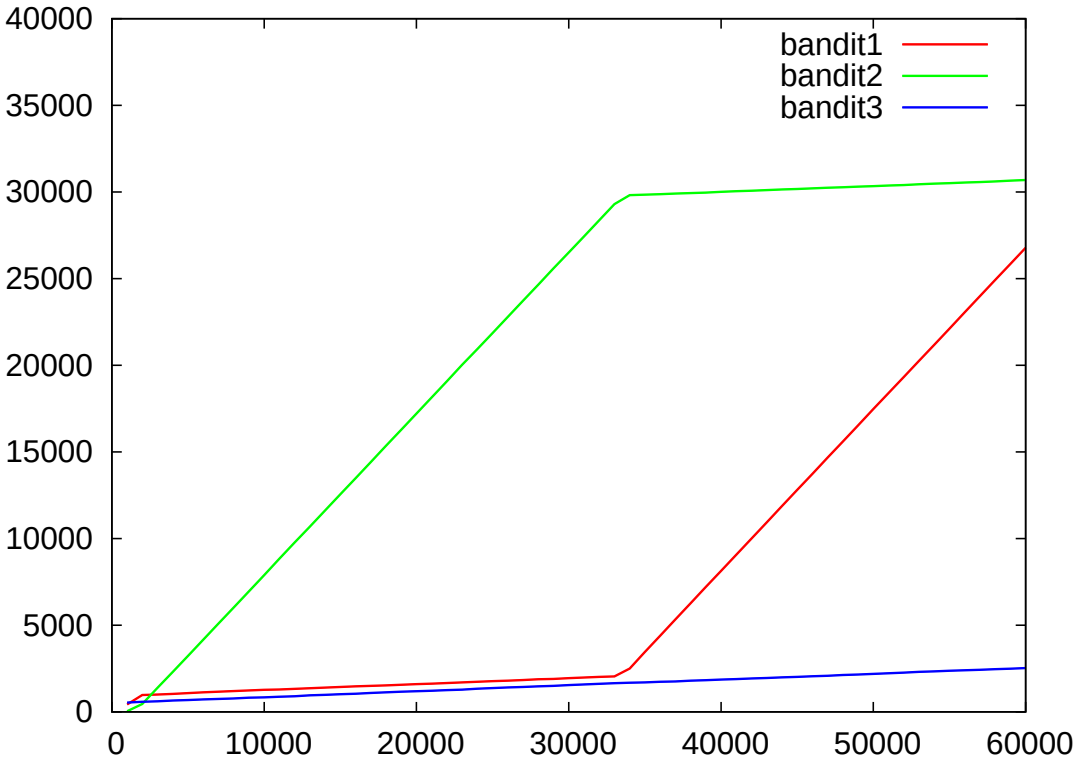
再びn本腕スロットマシン問題(continued)

- さっきの問題のスロットマシンを考える
- 計3万回レバーを押した後は、報酬は次のように変わる
  (エージェントはそのことを知らない)

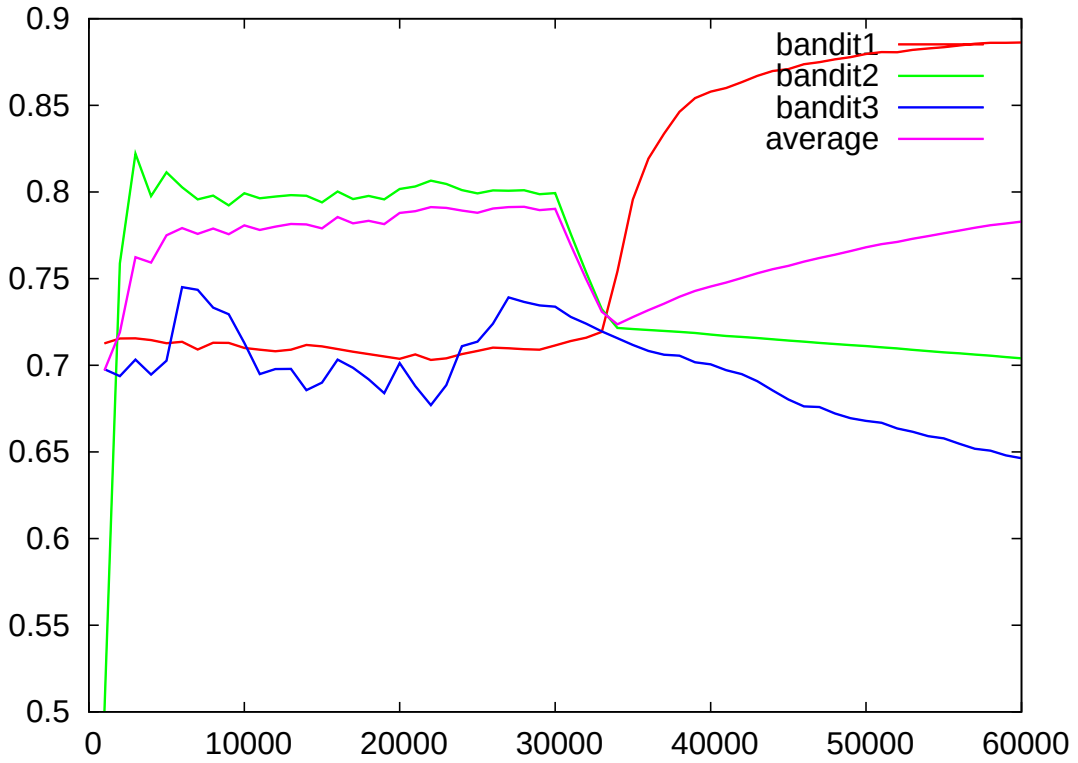
レバー1	レバー2	レバー3
確率0.9で報酬1	確率0.1で報酬1	確率0.5で報酬1
確率0.1で報酬0	確率0.0で報酬0	確率0.5で報酬0
平均報酬0.9	平均報酬0.1	平均報酬0.5

- ε-greedy法で試してみる

再びn本腕スロットマシン問題(continued)



各レバーを押した回数



各レバーの推定平均報酬と
全体の平均報酬

再びn本腕スロットマシン問題(continued)

- 最初の３万回は、推定平均報酬が最も良いレバー２が多く押される
- それを過ぎると推定平均報酬が変わっていって、そのうちレバー１の推定平均報酬が最も良くなり、レバー１が多く押されるようになる

推定平均報酬の更新

- あるレバーを今までn回試して平均報酬がrだったとする
- 今回もう1回試して報酬がiだったら、新たな平均報酬は

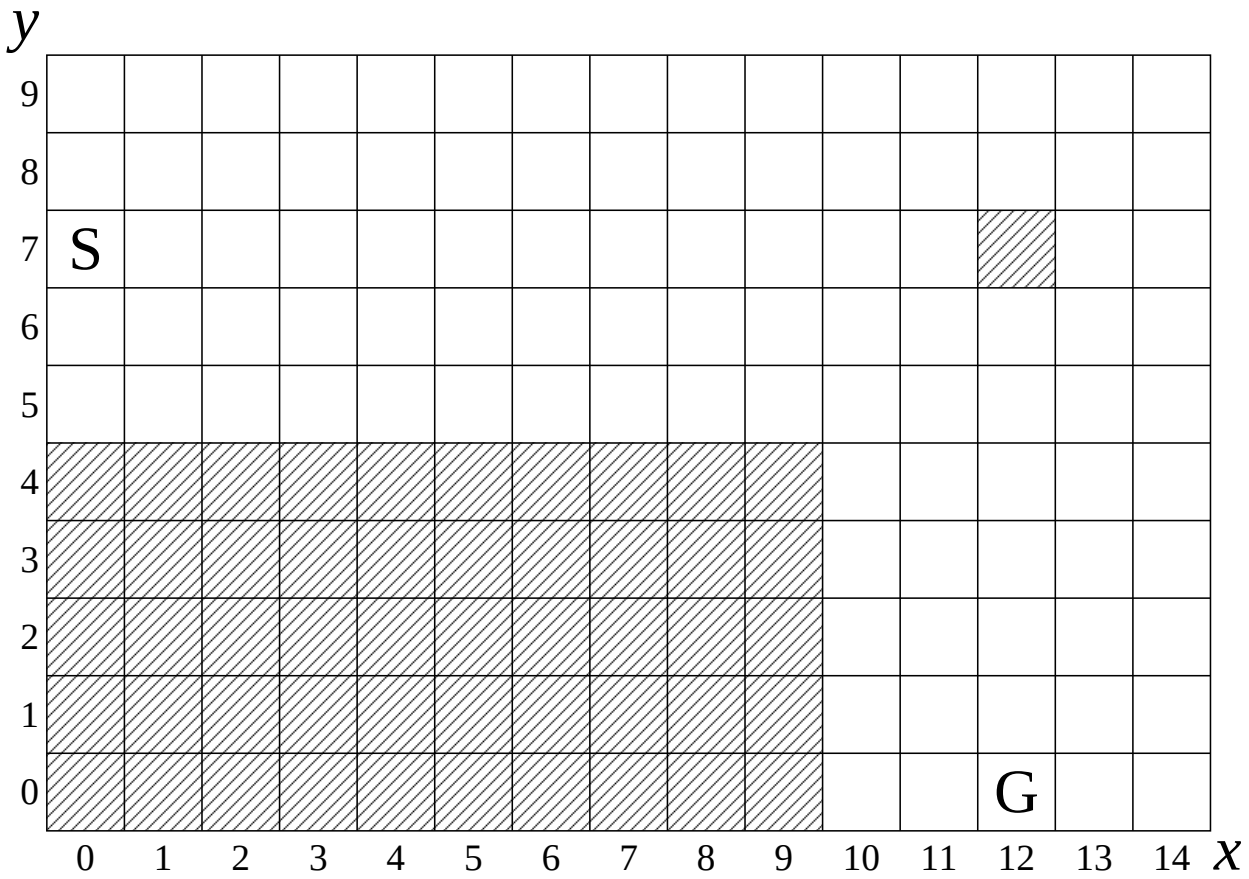
$$\frac{n \cdot r + i}{n+1} = \frac{(n+1)r + (i-r)}{n+1} = r + (i-r)\frac{1}{n+1}$$

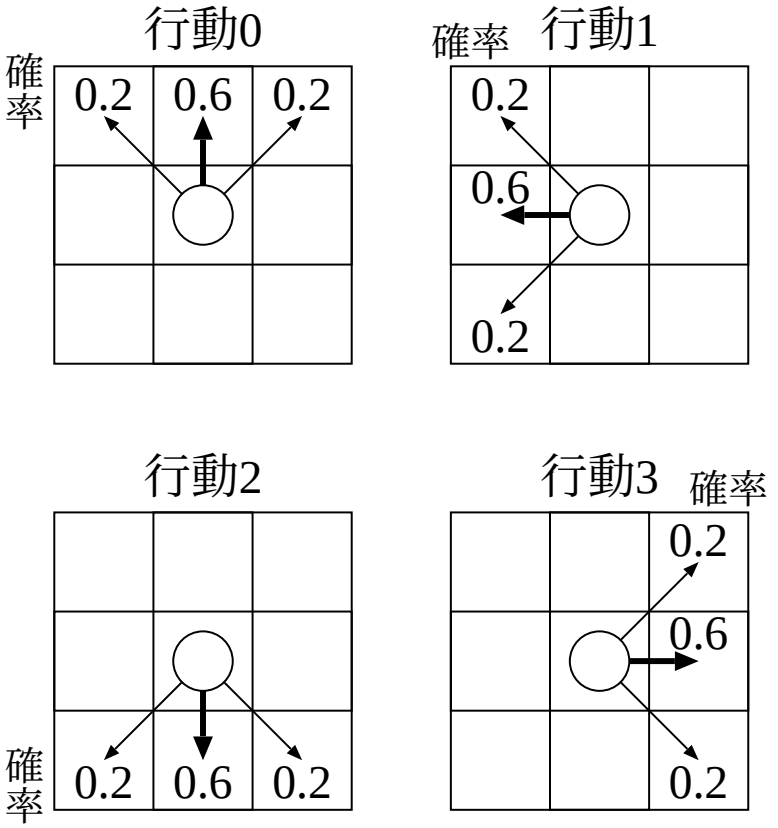
- つまり、rからiの方へ少し(iとrの差の$\frac{1}{n+1}$倍ほど)近づいた値
- 係数として$\frac{1}{n+1}$の代わりに小さな定数αを使うことがあり、これを「ステップサイズパラメータ」という

状態遷移がある問題での学習

- 実際には状態遷移をしながら試行を行う問題が多い
- 例題
  - ★ 盤の上をエージェントが動く
  - ★ 盤の1つ1つのマスを「状態」と考える
  - ★ エージェントの行動は4通りのいずれか
  - ★ エージェントの移動(状態遷移)は確率的
  - ★ エージェントの目的はゴールになるべく速く到達すること
  - ★ 盤の外や斜線のマスに入ったら「失敗」で終わり
  - ★ スタートからゴールに着くか失敗するかまでを「1エピソード」と呼ぶ

状態遷移がある問題での学習(continued)





- Sがスタート、Gがゴール
- エージェントは行動0を選ぶと確率0.6で上へ行く

状態遷移がある問題での学習(continued)

- 速い動き方を学習させるため、「報酬」を次のように設定
  - ⋆ Gに着いたら100
  - ⋆ 盤外か斜線のマスに入ったら-100
  - ⋆ それ以外なら移動1回につき-1
- 1エピソードでの総報酬をできるだけ大きくすることを目指す
- それぞれの状態(マス)でのそれぞれの行動(行動0～3)の価値を推定していく
  - ⋆ 状態sでの行動aの価値(状態行動価値)を$Q(s, a)$と書く

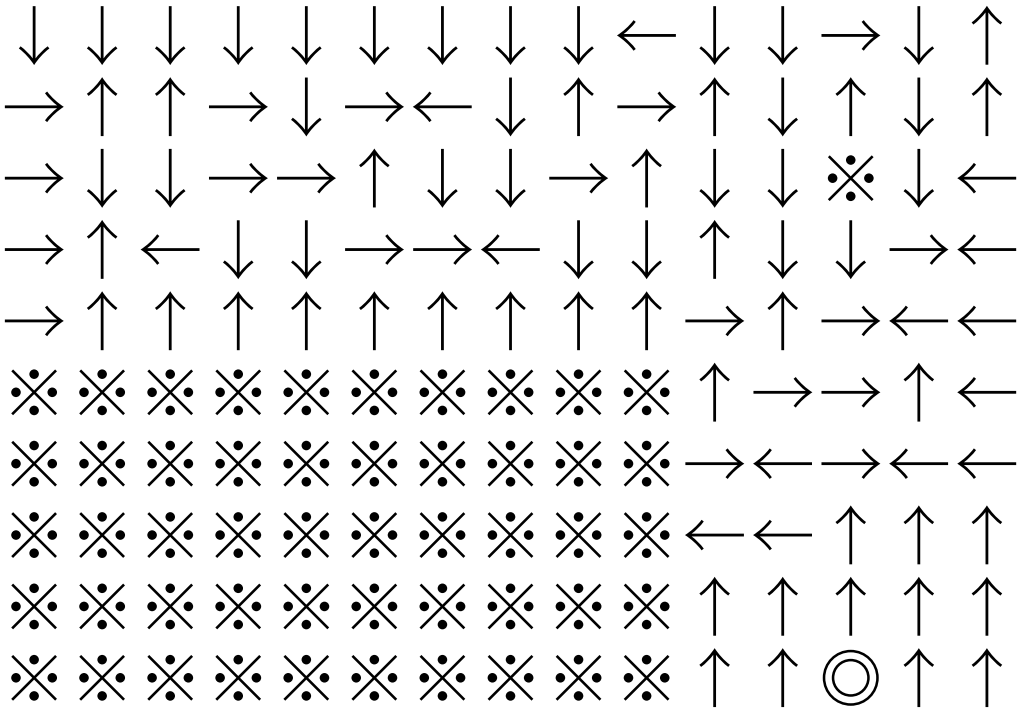
状態遷移がある問題での学習(continued)

- ⋆ 今回、状態sで行動aをとったとして、$Q(s,a)$を以下のように推定
  - ∗ aで移動した新しい状態s'での最善の行動(推定)をa'とする
  - ∗ aで今回得た報酬をrとする
  - ∗ $Q(s,a)$を$r+\gamma\times Q(s',a')$の方に少し近づける (γは1にするか、1より少しだけ小さい適当な定数にする。γは「割引率」と呼ばれる)
- ⋆ この学習方法は「Q学習」と呼ばれる

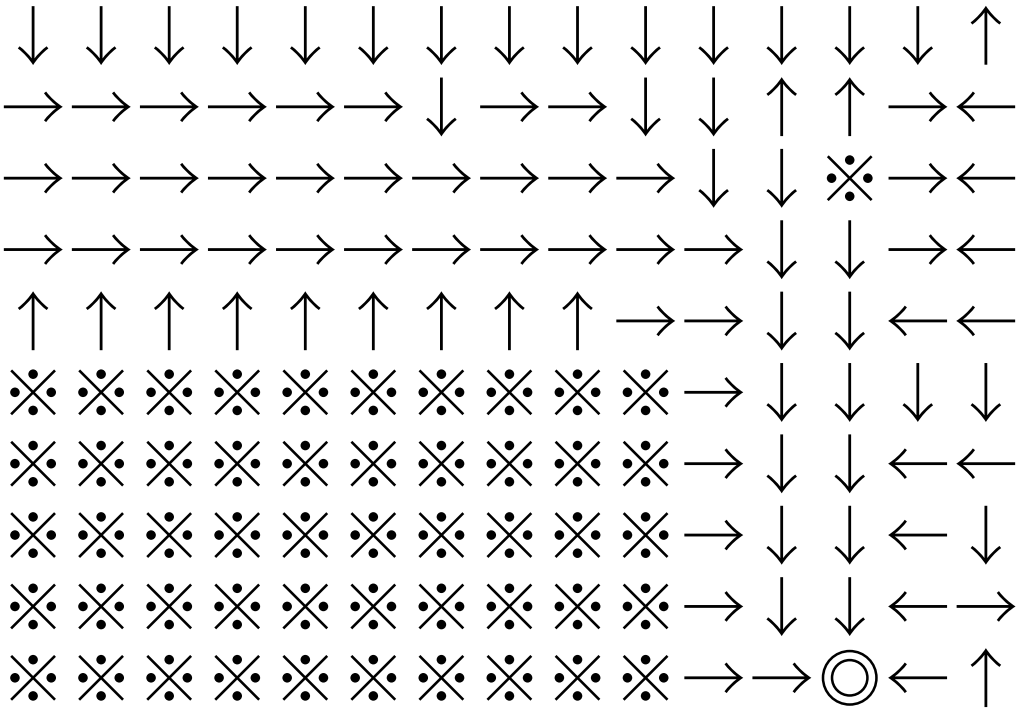
状態遷移がある問題での学習(continued)

- 学習結果
  - ★ 各状態での最善の行動の推測

100エピソード目



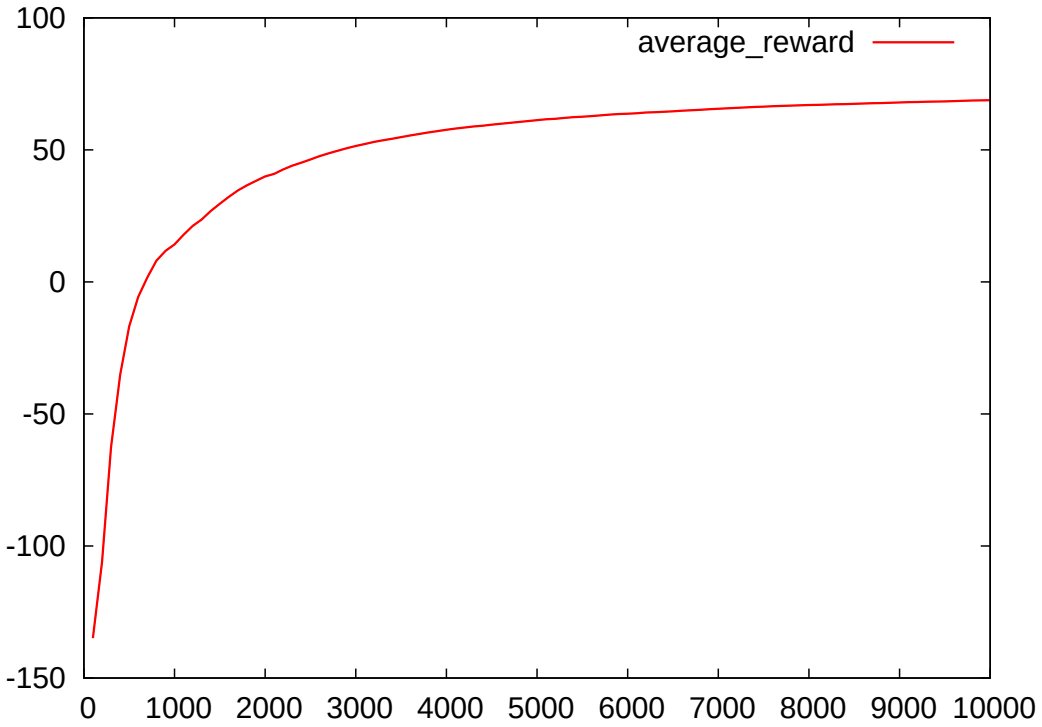
1000エピソード目



  - ★ 次第に、ゴールに近づく行動を最善とする推測に落ち着いていく

状態遷移がある問題での学習(continued)

- 1エピソードごとの総報酬の平均(横軸はエピソード数)



- 平均報酬は増大していくが、ε-greedyでときどき変な手をとるので、ある程度以上大きくならない

付録2: プランニング

プランニング

- 与えられたゴールを達成するプランを生成する問題
- 初期状態・ゴール・可能なアクション(オペレータ)の集合を与えて、初期状態からゴールへのアクションの列(プラン)を求める
- 2000年頃に盛んに研究され、TLPLANやSHOP2など効率のよいプランナがいくつか発表された

プランニング問題の例: ブロックワールド

- 状態の表現: 述語を用いる
  - ⋆ on(X,Y) · · · ブロックXはブロックYのすぐ上に乗っている
  - ⋆ on(X,table) · · · ブロックXはテーブルのすぐ上に乗っている
  - ⋆ clear(X) · · · ブロックXの上には何も乗っていない
  - ⋆ holding(X) · · · ロボットの腕はブロックXを持っている
- 初期状態: 最初の状態で満たされている述語の集合
- ゴール: 最終的に満たしたい述語の集合で表す

ブロックワールド(continued)

- オペレータは名前、前提条件、効果からなる
  - ⋆ pickup(X)　(ブロックXをテーブルから持ち上げる)
    前提条件･･･on(X,table), clear(X)
    効果･･･¬on(X,table), ¬clear(X), holding(X)
  - ⋆ putdown(X)　(ブロックXをテーブルに置く)
    前提条件･･･holding(X)
    効果･･･¬holding(X), on(X,table), clear(X)
  - ⋆ unstack(X,Y) (ブロックXをブロックYから持ち上げる)
    前提条件･･･on(X,Y), clear(X)
    効果･･･¬on(X,Y), ¬clear(X), holding(X), clear(Y)
  - ⋆ stack(X,Y)　(ブロックXをブロックYの上に置く)
    前提条件･･･holding(X), clear(Y)
    効果･･･¬holding(X), ¬clear(Y), on(X,Y), clear(X)

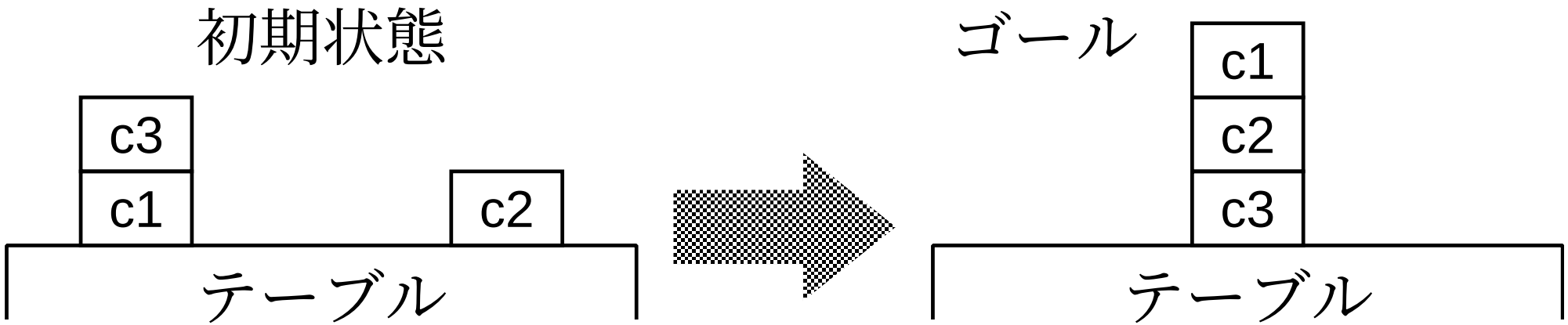
ブロックワールド(continued)

- 初期状態

  on(c1,table), on(c3,c1), clear(c3), on(c2,table), clear(c2)

- ゴール

  on(c1,c2), on(c2, c3)



答･･･unstack(c3,c1), putdown(c3), pickup(c2), stack(c2,c3), pickup(c1), stack(c1,c2)

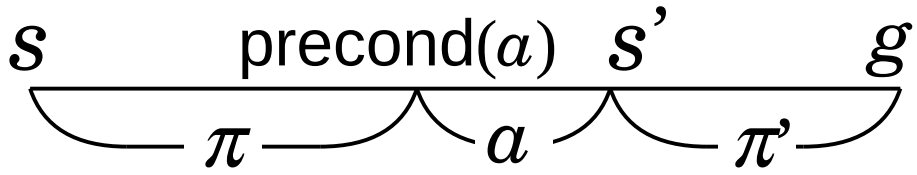
STRIPSアルゴリズム

オペレータの集合Oがあるとき、状態s, ゴールgを与えて、sからgへOのオペレータの列で行けるプランを生成

sがgを満たすなら空のプランを返す
gに寄与するグラウンドオペレータaを1つ選ぶ
　(そのようなものが多数あるならそのそれぞれについて以降を行う)
sからaの前提条件へ行けるプランπを生成(なければ失敗)
sからπおよびaで行ける状態をs'とする
s'からgへ行けるプランπ'を生成(なければ失敗)
π, a, π'をつなげればそれが答

「グラウンド」とは変数が代入済みであること。「aがgに寄与する」とは、aで成り立つようになるgの要素が1つ以上あり、aで成り立たなくなるgの要素がないこと

STRIPSアルゴリズム(continued)



- STRIPSアルゴリズムは初期のプランナの研究において重要な成果(1971年)
- 従来のプランナより効率を改善
- ただし、出てくる答に無駄な部分が含まれる例外的な事象(Sussmanの例外)も知られる

STRIPSアルゴリズム(continued)

- 今回の例では
  - ゴールに寄与するグラウンドオペレータstack(c1,c2)を選ぶ
  - 初期状態からstack(c1,c2)の前提条件を達成するプランは

    unstack(c3,c1), putdown(c3), pickup(c1)
  - このプラン＋stack(c1,c2)で行ける状態からゴールに行くプランを別途作る

    unstack(c1,c2), putdown(c1), pickup(c2),
    stack(c2,c3), pickup(c1), stack(c1,c2)
- 両者をつなげれば答(しかしこの解はSussmanの例外を起こしている)