知能システム学
第3回　問題解決、
問題のモデル化

ソフトウェア情報学部
David Ramamonjisoa

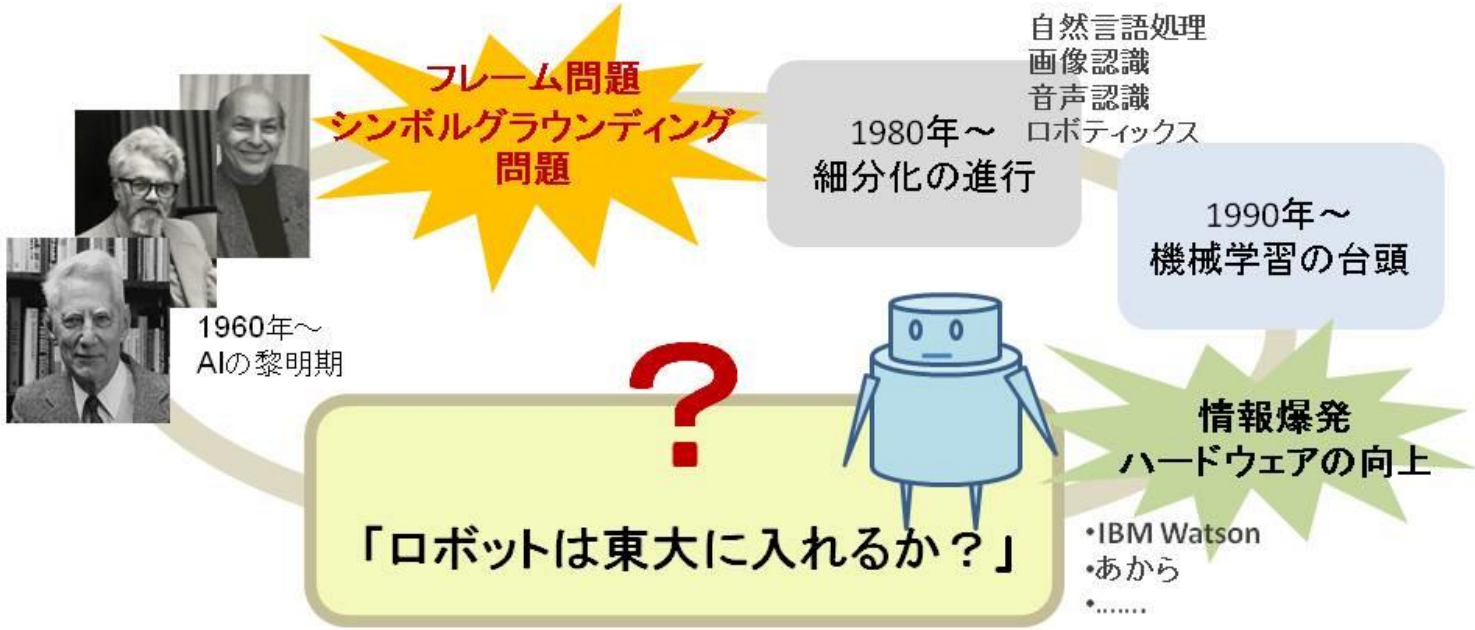
目次

- 問題の種類
- 問題解決プロセス？
- 問題解決エージェント
- タスク環境の設定
- 問題を定式化すること
- 例題
- まとめ

問題の種類(The Problem types)

- おもちゃの問題 (Toy problem)
  - パズル、ルビクス　キュブ、数独、など
  - ゲーム:チェス、囲碁、ポカー、ブリッジ、...
  - N-queens 、など
- 現実世界問題 (Real-world problem)
  - 文章題(文章の形式になっている問題)- 数学
  - ルート設計や発見問題
  - 環境問題(気候変化、災害、人口、など)、政治問題(政権、憲法、民法、税金、防衛、など)、化学技術問題(ロボット操縦、タンパク質設計、薬剤発見、バイオテクノロジー、社会ネット、インターネット検索、など)

現実世界問題の例：東大ロボットプロジェクト（21robot.org）

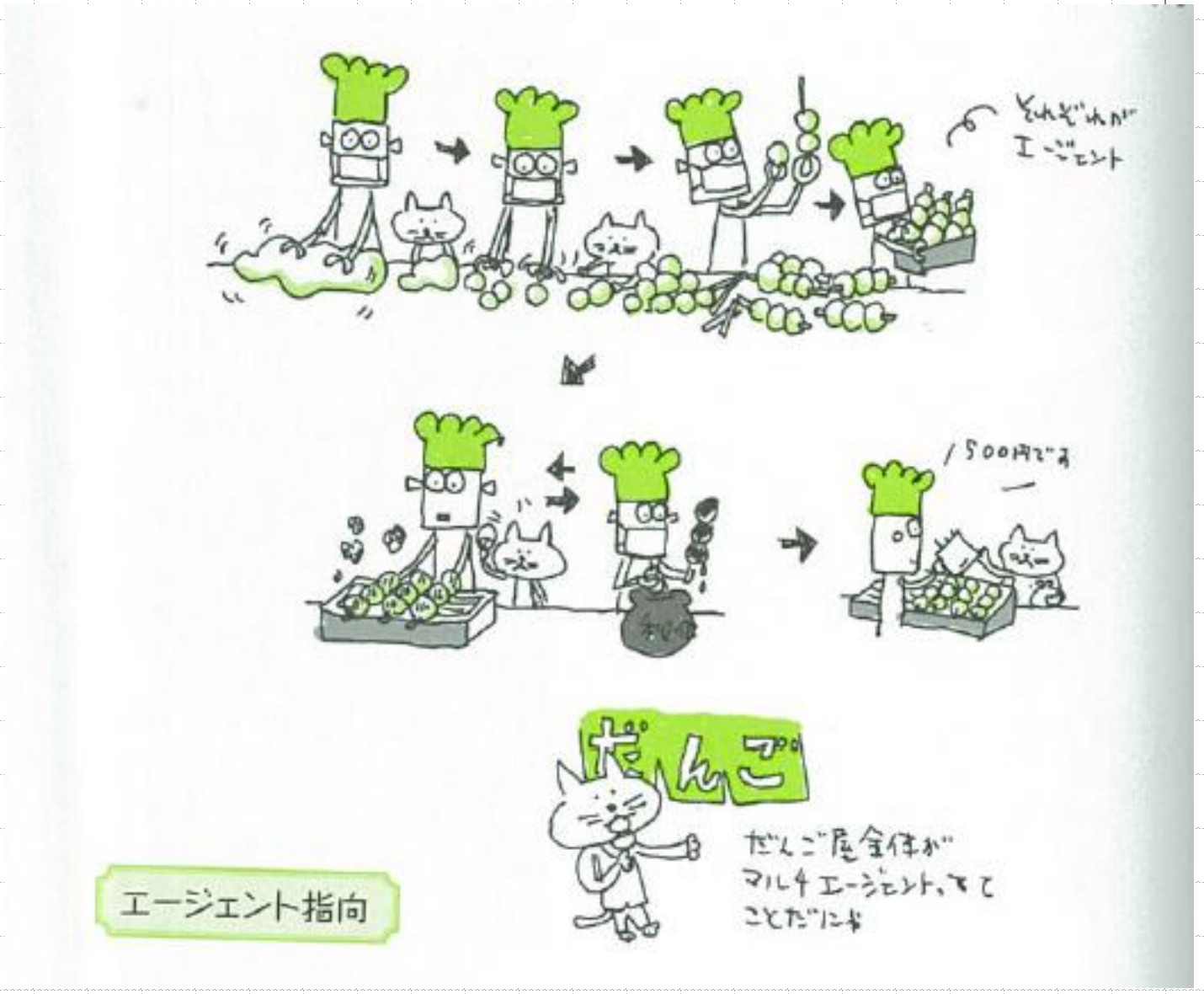


問題解決とは？

- 問題発見のあとに来る、解決手段を探り、実際に解決に至る道筋を示す行為。
- 典型的な問題は、いくつかの解決手段が用意されている。
  - 診断問題
  - 計画問題
  - スケジューリング問題
  - 設計問題
  - 予測問題
- 現実の問題は、複合問題となっていることが多く、問題の整理、形式化が重要。
- 典型的な問題は、問題解決エージェントの枠組みで整理できる。

問題解決エージェント

- 問題解決エージェントは，ゴールが与えられて，そのゴールに至る解を探索する．
- はじめに，ゴールを定式化する．
  - **ゴール**は，**世界状態の集合**として定義される．
  - **行為**は，**世界状態間の遷移**を引き起こす．
- つぎに，問題の定式化を行う．
  - 適切な行為（オペレータ）の選択が必要．
- つぎに，解に至る行為の様々な可能な系列を調べて，その中で最も良いものを選ぶ（解の探索）．
- 最後に，解を実行する．


それぞれがエージェント
1500円でする
だんご
だんご屋全体がマルチエージェントってことだにゃ
エージェント指向

人工知能（AI）の問題

- 弱いAIと強いAI
  - 弱いAI：単独な専門家の知識や技能を達成する

ゲーム専門家（チェス、囲碁、将棋など）、診断者、自動運転、自然言語の認識など

  - 強いAI：人間のように知識を発見し、創造し、自己意識などの能力（マルチタレント）

エージェントが自分が問題を見つけて、解決する

人工汎用知能（AGI）またはシンギュラリティ（Singularity）：2045年の問題

自動運転のレベル分けについて

国土交通省
【別添３】

システムによる監視

ドライバーによる監視

高速道路での完全自動運転（2025年目途）

限定地域での無人自動運転移動サービス（2020年まで）

高速道路等一定条件下での自動運転モード機能を有する「自動パイロット」（2020年目途）

レベル5

レベル4

レベル3

レベル2

レベル1

○完全自動運転
常にシステムが全ての運転タスクを実施

○特定条件下における完全自動運転
特定条件下においてシステムが全ての運転タスクを実施

*6

○条件付自動運転
システムが全ての運転タスクを実施するが、システムの介入要求等に対してドライバーが適切に対応することが必要

*5

○特定条件下での自動運転機能（高機能化）
【例】 高速道路での自動運転モード機能
①遅いクルマがいれば自動で追い越す
②高速道路の分合流を自動で行う

*4

○特定条件下での自動運転機能（レベル１の組み合わせ）
【例】車線を維持しながら前のクルマに付いて走る（LKAS+ACC）

○運転支援　システムが前後・左右のいずれかの車両制御を実施

【例】自動で止まる（自動ブレーキ）

*1

前のクルマに付いて走る（ACC）

*2

車線からはみ出さない（LKAS）

*3

ACC：Adaptive Cruise Control，LKAS: Lane Keep Assist System

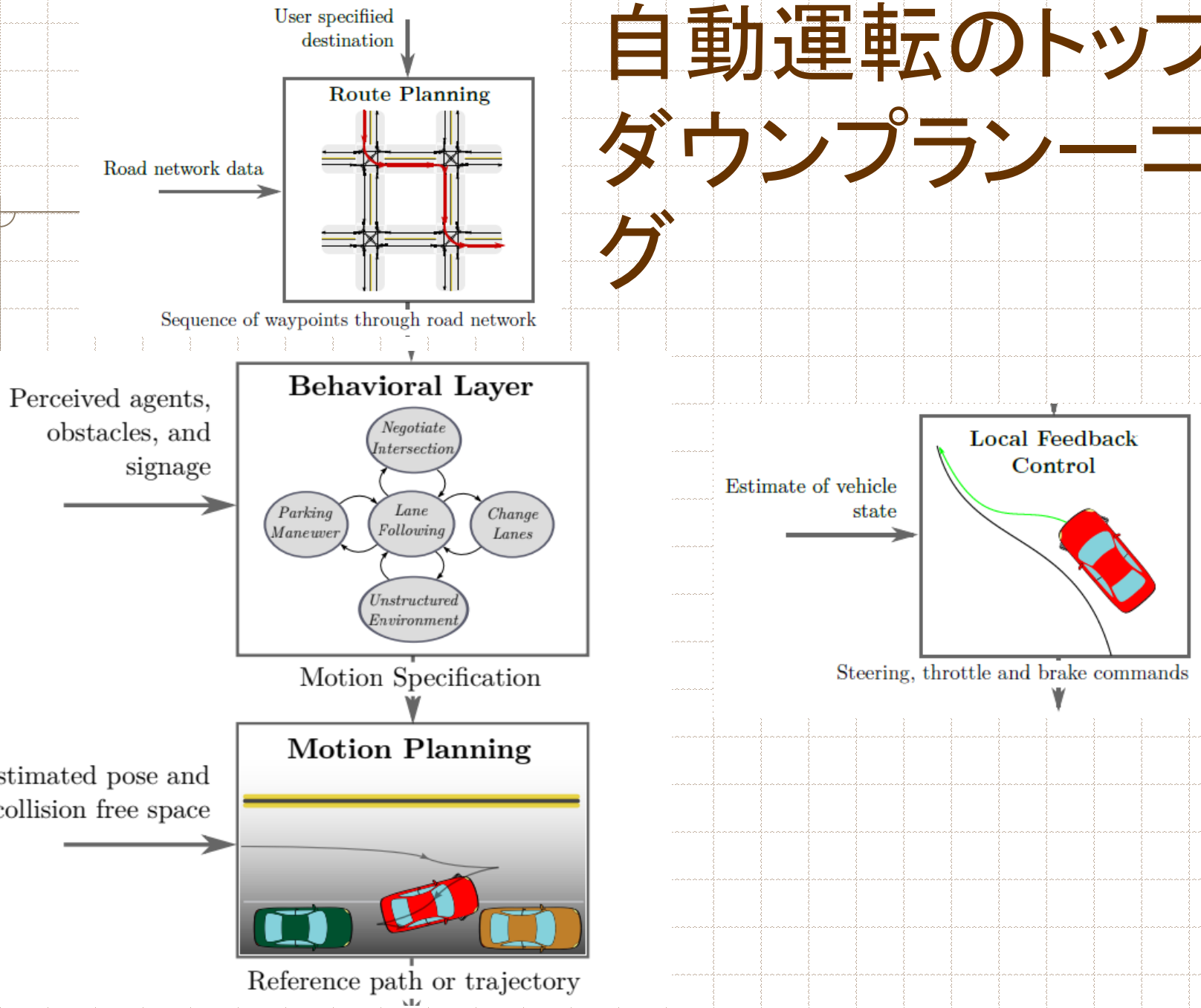
官民ITS構想・ロードマップ2017等を基に作成

*1（株）SUBARUホームページ　*2 日産自動車（株）ホームページ　*3 本田技研工業（株）ホームページ
*4 トヨタ自動車（株）ホームページ　*5 Volvo Car Corp.ホームページ　*6 CNET JAPANホームページ

自動運転のタスクプランーニング

- 出発位置から目的位置までの最適な道路検索
- 様々な場面で問題解決（最適な経路、無事故、快適な走行、詳細計画、…）
- 行為：追い起し、車線追跡、車線変更、駐車交差点の交渉、交通ルールの尊重、障害物回避、安全運転

自動運転のトップダウンプランーニング



機械学習の問題解決法

- データ収集
  - ラベル付事例、整理
- モデル構築:
  - 構成設定、最適基準、自動改定
- モデル評価
- オンライン利用

この授業で対象の問題：弱いAI

- 問題を定義する
  - 初期状況、環境
  - 終了状況、評価関数
- ゴールを設定し、ゴールを達成する。

タスク環境(task environment)

- エージェントを設計する際に性能指標と環境、アクチュエータ、センサーの使用を決めなければならない。
- これかの使用項目はPEAS（Performance:性能、Environment:環境、Actuators:アクチュエータ、Sensors:センサー ）表現とよぶことにする。

PEAS

- エージェントの設計では、最初のステップでは必ず可能な限り詳細にタスク環境を設計しなければならない。
- 例：自動タクシーに関するタスク環境のPEAS表現
  - 性能指標: 安全(Safe), 速い(fast), 偽物ではない(legal), 快適性(comfortable trip), 利益を最大化(maximize profits)
  - 環境: 道路(Roads), 他の車、歩行者など(other traffic), 見込み客(customers)
  - アクチュエータ:アクセル、ハンドル、ブレーキなど(Steering wheel, accelerator, brake), 乗客とのコミュニケーションツール、など
  - センサー: カメラ(Cameras), ソナー(sonar),スピードメーター（speedometer), GPS, odometer, engine sensors, keyboard

PEAS

- エージェント: 医療診断システム(Medical diagnosis system)
  - 性能指標: 元気(Healthy patient), 最小コスト(minimize costs), 法律(lawsuits)
  - 環境: 患者(Patient), 病院(hospital),職員(staff)
  - アクチュエータ: ディスプレイ(Screen display) (questions, tests, diagnoses, treatments, referrals)
  - センサー: キーボード(Keyboard)(データの入力(entry of symptoms, findings, patient's answers))

PEAS

- エージェント: ベルトコンベヤによって運ばれてくる物品を検査するロボット(Part-picking robot)
- 性能指標: 正しい物品を置き場に入るのパーセンテージ(Percentage of parts in correct bins)
- 環境:ベルトコンベヤーと 物品(Conveyor belt with parts),置き場(bins)
- アクチュエータ: 腕、手(Jointed arm and hand)
- センサー：カメラ(Camera), センサー(joint angle sensors)

PEAS

- エージェント: オンライン対話英語教師 (Interactive English tutor)
- 性能指標: テストの評価を最大化する (Maximize student's score on test)
- 環境: 生徒達　(Set of students)
- アクチュエータ: ディスプレイ(exercises, suggestions, corrections)
- センサー：キーボードの入力

タスク環境と性質

- 完全観測可能　対　部分観測可能
- 決定的　対　確率的
- 挿話的　対　系列的
- 静的　対　動的
- 離散的　対　連続的
- 独立　対　マルチ
- 競争的、協調的

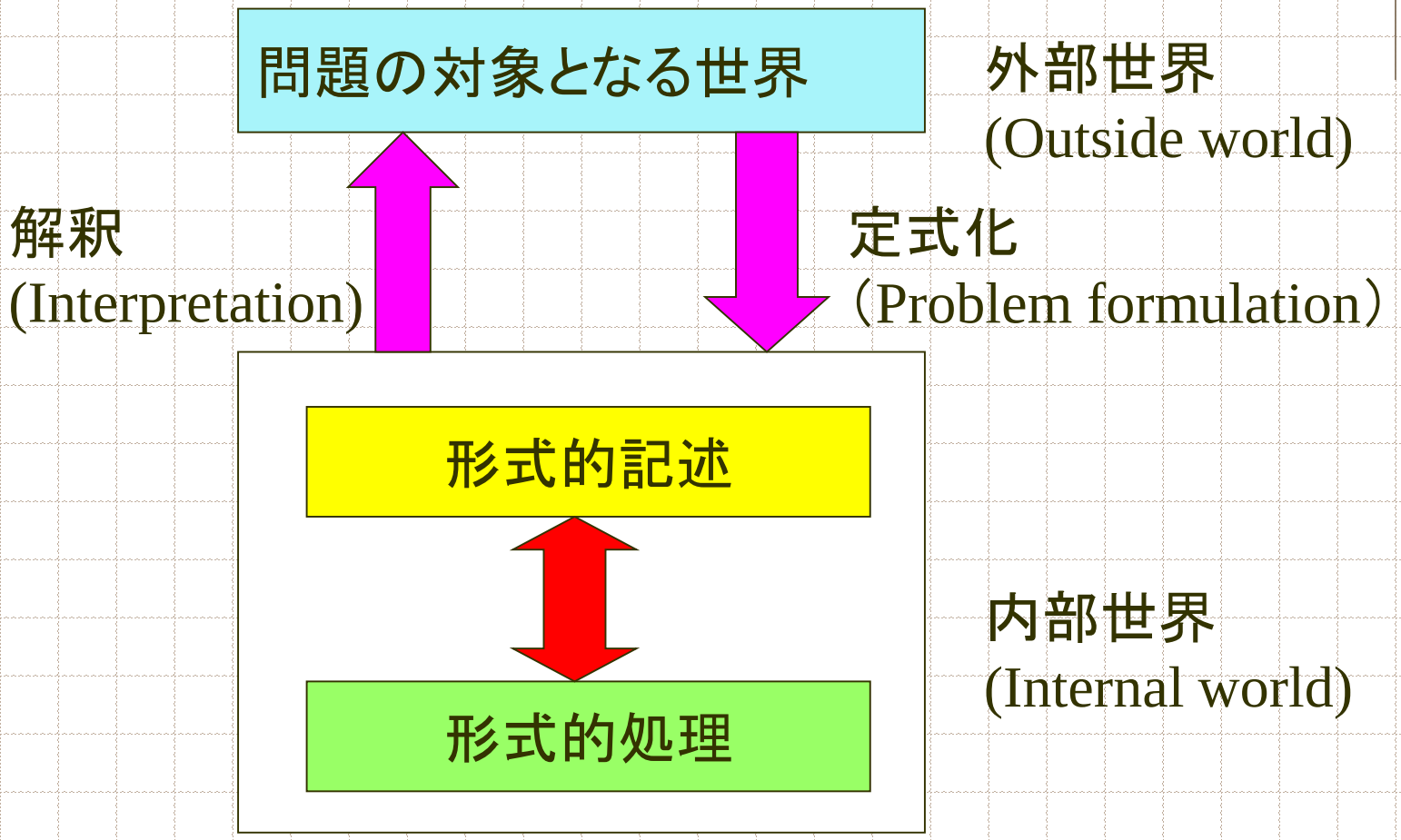
問題解決エージェント(Problem-solving Agents) (2)

- ゴールの定式化:
  - 鶴と亀の数を知る
  - 自転車とバイクの数を知る
- 問題の定式化:
  - 日本語で書かれた問題の内容から連立方程式を立てる
- 形式的記法:
  - 連立方程式で記述する
- 形式的処理:
  - 連立方程式を解ける(式の変形、逆行列の計算など)

問題解決プロセス(The Problem Solving Process)

- 鶴と亀の数の問題
  - 鶴と亀が合計7匹いる
  - 鶴と亀の足の合計は20本である。
  - このとき、鶴と亀は各々何匹いるのてしょう。
- 車とバイクの数の問題
  - 車とバイクが合計9台いる
  - 車とバイクのタイヤの合計は30本である。
  - このとき、車とバイクは各々何台あるのてしょう。

問題解決プロセス（The Problem Solving Process）



問題解決エージェント(Problem-solving Agents) (2)

- **問題の定式化:**
  - 未知数を用いて、各々を $x,\ y$ とする。
- **形式的記法:**
  - 連立方程式で記述する

$$x+y=7,\ 2x+4y=20$$

- **形式的処理:**
  - 連立方程式を解ける(式の変形、逆行列の計算など)

$$\begin{cases} x+y=7 \\ 2x+4y=20 \end{cases} \quad \begin{cases} 2x+2y=2*7 \\ 2x+4y=20 \end{cases} \quad \begin{cases} 2y=20-14=6 \\ 2x+4y=20 \end{cases}$$

$$y=3,\ \ x+3=7 \Leftrightarrow x=4, y=3$$

問題解決エージェント(Problem-solving Agents) (2)

- **解釈:**
  - 内部世界の結果を外部世界の表現で表す

$$x = 4,\ y = 3$$

- 「**鶴が4匹、亀が3匹**」

- **2番目の問題は同様に解決する**

問題解決

- 「誰かが負けない数の平方根を探すプログラムを書いてほしいとお願いしてきたとしよう。どうすべきだろうか?」
- 問題の定式化
  - xは入力の数、yは見つけたい数
  - 条件：x = y*y or x – y*y <= Epsilon, Epsilon 小さい値
  - ゴール：平方根のおおよその値を見つけた（ある範囲内＝Epsilon）

問題解決：平方根の近似解

```
# ２分法
ans=1.0
x=24
epsilon=0.01

numLoop=0
while abs(ans*ans- x)>= epsilon:
    ans = (ans + x/ans)/2
    print ('ans =',ans,' and num loop =', numLoop
    numLoop += 1
print ('num loop = ', numLoop)

print ('Answer is = ', ans)
```

```
# Newton-Raphson's method

x=24
epsilon=0.01
ans=x/2.0
numLoop=0
while abs(ans*ans- x)>= epsilon:
    ans = ans - (ans*ans - x) /(2*ans)
    print ('ans =',ans,' and num loop =', numLoop)
    numLoop += 1
print ('num loop = ', numLoop)

print ('Answer is about : ', ans)
```

16\知能システム学I 2016\p1.py ==========

```
('ans =', 12.5, ' and num loop =', 0)
('ans =', 7.21, ' and num loop =', 1)
('ans =', 5.2693550624133145, ' and num loop =', 2)
('ans =', 4.911996075481051, ' and num loop =', 3)
('ans =', 4.898996732283415, ' and num loop =', 4)
('num loop = ', 5)
('Answer is = ', 4.898996732283415)
```

16/知能システム学I 2016/p2.py ==========

```
('ans =', 7.0, ' and num loop =', 0)
('ans =', 5.214285714285714, ' and num loop =', 1)
('ans =', 4.908512720156556, ' and num loop =', 2)
('ans =', 4.8989887432139305, ' and num loop =', 3)
('num loop = ', 4)
('Answer is about : ', 4.8989887432139305)
```

例えば、x=351363189のとき、yの値、ループの数はどのくらい

問題解決エージェント(Problem-solving Agents) (3)

```
function Simple-Problem-Solving-Agent(p) returns an action
    inputs:     p, percept
    static:     s, an action sequence, initially empty
                state, some description of the current world state
                g, a goal, initially null
                problem, a problem formulation
    state ← Update-State(state, p)
    if s is empty then
        g ← Formulate-Goal(state)
        problem ← Formulate-Problem(state, g)
        s ← Search(problem)
    action ← Recommendation(s, state)     /* first action in the sequence
    s ← Remainder(s, state)               /* other actions in the sequence
    return action
```

対象とする問題

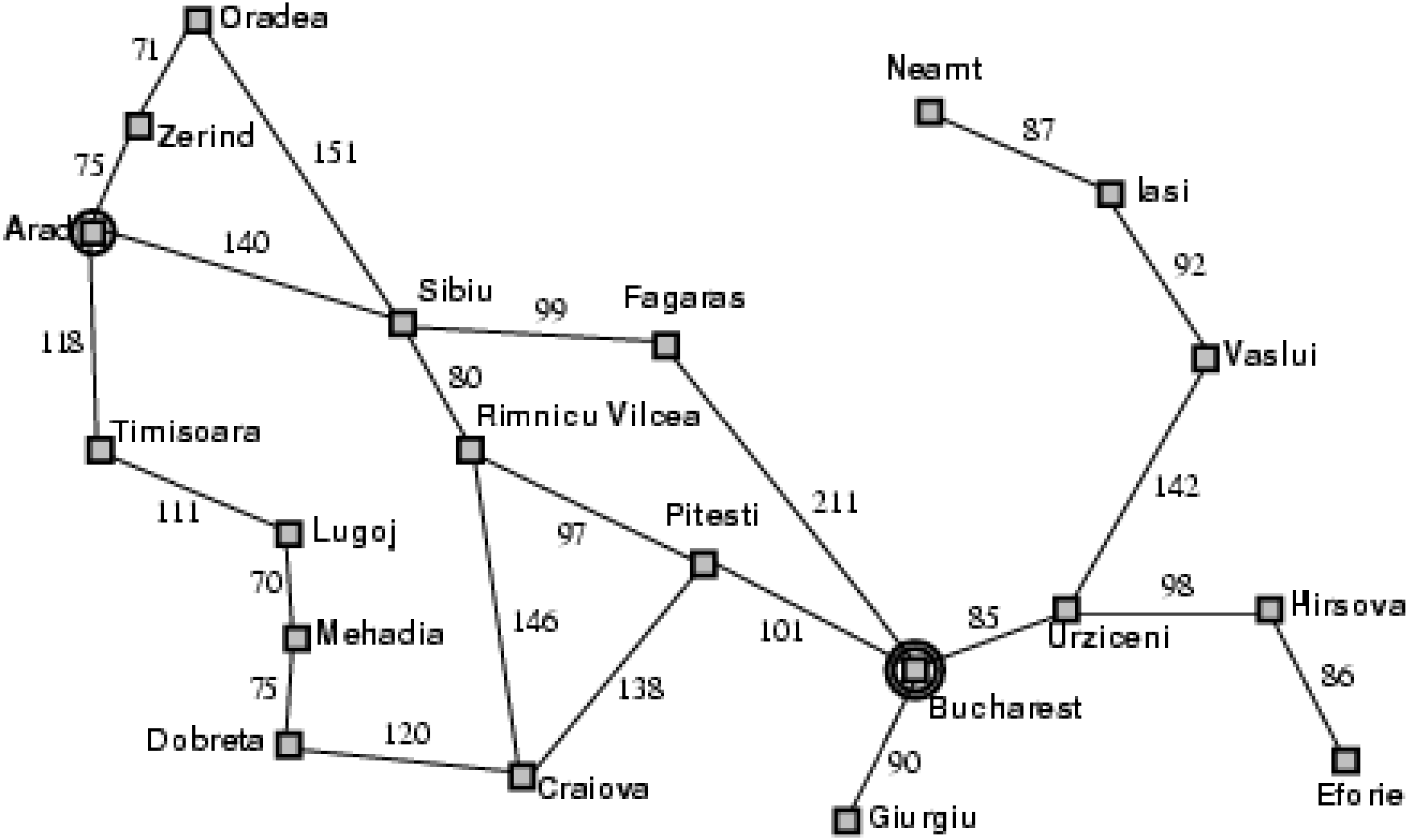
- 人間の知的活動を工学的に実現する
- 問題発見のあとに来る、解決手段を探り、実際に解決に至る道筋を示す行為。
- 典型的な問題は、いくつかの解決手段が用意されている。
  - 診断問題
  - 計画問題
  - スケジューリング問題
  - 設計問題
  - 予測問題
- 現実の問題は、複合問題となっていることが多く、問題の整理、形式化が重要。
- 典型的な問題は、問題解決エージェントの枠組みで整理できる。

問題解決エージェントの例

- 旅行者（エージェント）が，ルーマニアの都市Aradに滞在している．エージェントは，次の日Bucharestから飛び立つチケットを持っている．どうすればよいか．
  - **ゴールの定式化**：
    - 「ドライブしてBucharestに行く」を，ゴールとして設定すべきである．
  - **問題の定式化**：
    - 「左足を前方に18インチ動かす」あるいは「ハンドルを左へ6度回す」などは，行為としては不適切である．
    - 都市間をドライブするというレベルの行為を考える．
  - **解の探索**：
    - 都市間の隣接情報から，AradからBucharestに行く可能なコースを探索し，最適なプランを立てる．
  - **解の実行**：
    - 実際に解に従って，AradからBucharestにドライブする．


SLOVAKIA
Košice
Uzhhorod
Mukachëve
Ivano-Frankivs'k
UKRAINE
Kam''yanets'-Podil's'kyy
Kolomyya
Prut
Dnistr
Kel'mentsi
Mohyliv-Podil's'kyy
Uman
Chernivtsi
Miskolc
Nyíregyháza
Tisza
BOTOŞANI
Botoşani
Bălți
Ribnița
Broshtyany
Satu Mare
SATU MARE
Carei
Baia Mare
MARAMUREŞ
Suceava
SUCEAVA
HUNGARY
Debrecen
Szolnok
BISTRIȚA-NĂSĂUD
IAŞI
MOLDOVA
Ungheni
Oradea
SĂLAJ
Zalău
Dej
Bistrița
NEAMȚ
Iași
Chisinău
BIHOR
CLUJ
Roman
Békéscsaba
Cluj-Napoca
Piatra-Neamț
Tighina
Vaslui
Turda
Târgu Mureș
MUREŞ
HARGHITA
Szeged
ARAD
Miercurea-Ciuc
BACĂU
Bacău
VASLUI
ALBA
Alba Iulia
Arad
Birlad
Mediaș
Artsyz
COVASNA
Timişoara
Deva
SIBIU
Sibiu
BRAŞOV
Sfîntu Gheorghe
VRANCEA
GALAȚI
Cahul
UKRAINE
Hunedoara
TIMIŞ
HUNEDOARA
Focșani
Vojvodina
(autonomous province)
Braşov
Galați
Reni
Kiliya
Izmayil
Danube
VÎLCEA
Petroşani
Reşiţa
BUZĂU
Brăila
Tulcea
Târgu-Jiu
GORJ
Rîmnicu Vîlcea
ARGEŞ
PRAHOVA
Buzău
BRĂILA
TULCEA
CARAŞ-SEVERIN
Drobeta Turnu Severin
Belgrade
Danube
Pitești
Târgoviște
Ploiești
DÎMBOVIȚA
IALOMIȚA
Smederevo
BUCUREŞTI
Slobozia
MEHEDINȚI
Slatina
Bucharest
Craiova
OLT
CĂLĂRAȘI
Călărași
Kragujevac
DOLJ
GIURGIU
CONSTANȚA
Constanța
Vidin
TELEORMAN
Silistra
Kraljevo
Calafat
Alexandria
Giurgiu
Ruse
Black Sea
Kruševac
Dobrich
Serbia
Niš
Montana
Pleven
Vratsa
Leskovac
BULGARIA
Kosovo
(autonomous province)
Priština
Pernik
Sofia
Uroševac
Romania
International boundary
County (județ) boundary
National capital
County (județ) center
Railroad
Expressway
Road

例: ロマニア(Romania)



問題を定式化すること

- 問題の構成要素
  - **初期状態**：エージェントが認識している最初の状態
  - **オペレータ**：エージェントが利用できる可能な行為
  - **ゴール検査**：エージェントが到達した状態がゴール状態であるかを決定する検査.
  - **経路コスト**：経路をたどるのに必要なコスト．経路コストは，経路に沿った各々の行為のコストの総和.
- 問題の環境
  - 状態空間：初期状態から行為の任意の系列によって到達可能なすべての状態の集合
  - 経路：1つの状態を他の状態に導く行為列

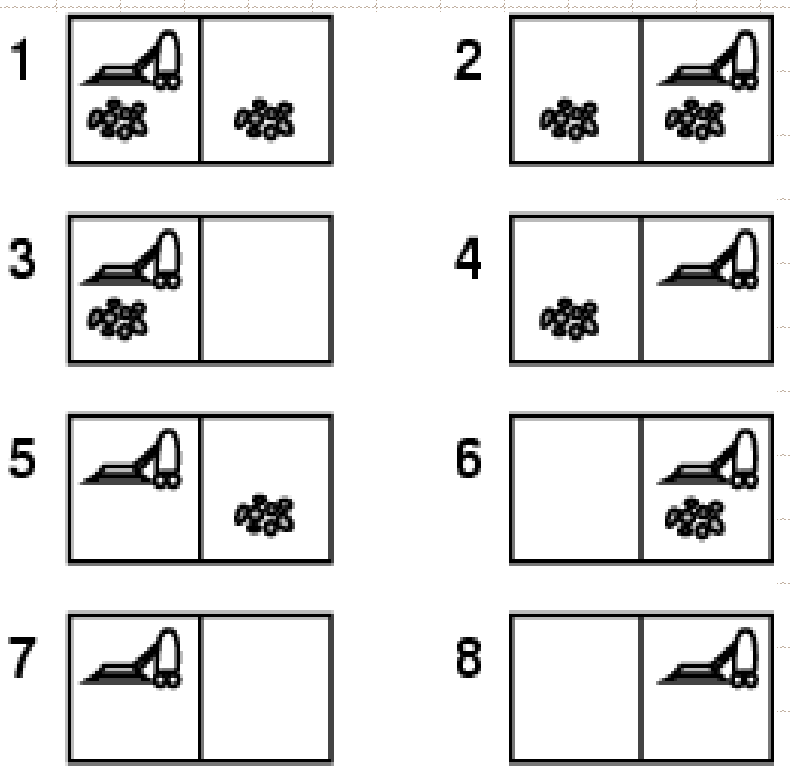
問題解決エージェントの例2 (掃除機の世界：vacuum world)

- 掃除機（エージェント）の世界は2つの場所だけを含むようにしよう．エージェントは、1つの場所か他の場所にいる可能性がある。8つの可能な正解状態がある。
  - **ゴールの定式化**：
    - 「**すべての埃をきれいに掃除することである**」を，ゴールとして設定すべきである．
  - **問題の定式化**：
    - 「**左へ移動（Left)**」あるいは「**右へ移動（Right)**」、「**吸込み（Suck)**」は，行為としてある．
  - **解の探索**：
    - 場所間の隣接情報から，可能な場所に行くし，**掃除する**．
  - **初期状態**：
    - どんな状態でも**OK**

問題解決エージェントの例2 (掃除機ロボットの世界：vacuum world)

- 単一状態問題(single-state problem)
- 単一状態(Single-state), 始点 #5.

解(Solution)?



掃除機の世界

- 単一状態(Single-state),

始点 #5.

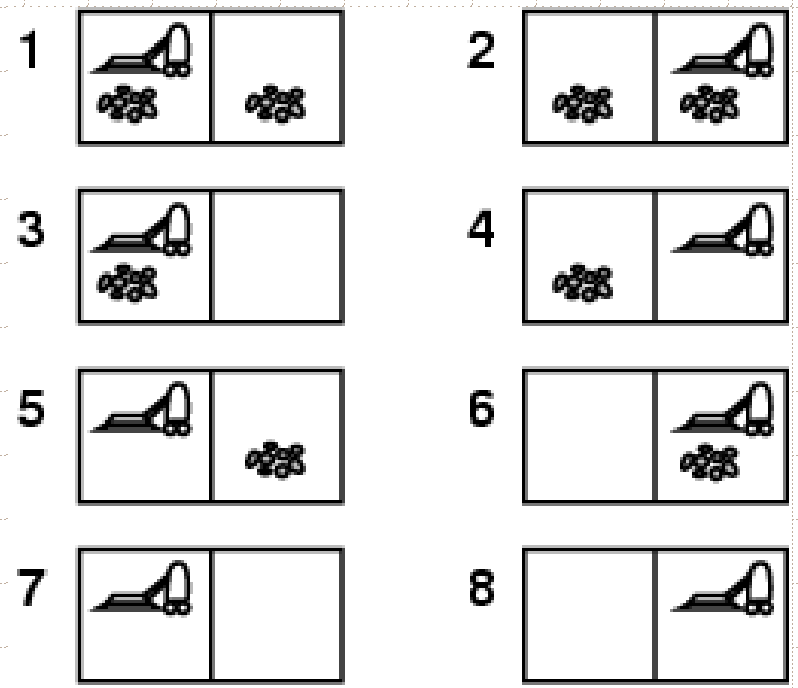
解? *[Right, Suck]*

- センサない（Sensorless）,

始点

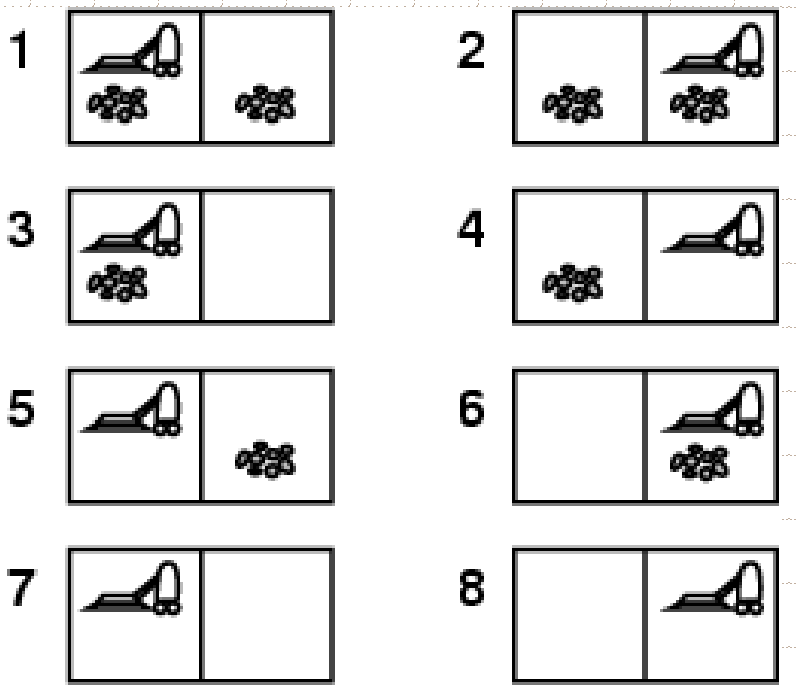
{*1,2,3,4,5,6,7,8*} e.g., *Right* なら {*2,4,6,8*}

解?



掃除機の世界

- センサない(Sensorless), 始点 {*1,2,3,4,5,6,7,8*} e.g., *Right*（右）なら {*2,4,6,8*}
  解?
  [Right,Suck,Left,Suck]

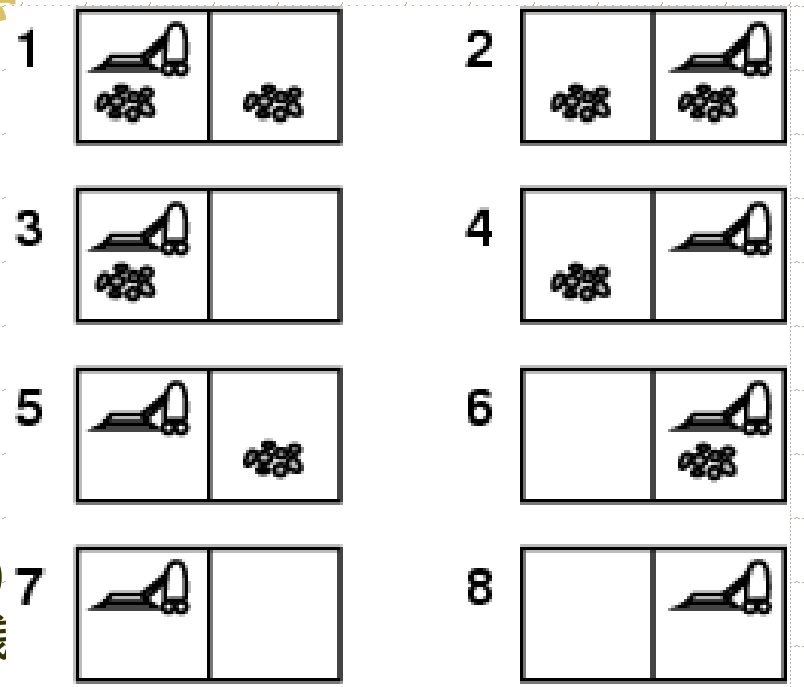


- 偶発的問題(Contingency)
  - 非決定論的(Nondeterministic): 吸い込みはきれいな部屋に埃を残す (*Suck* may dirty a clean carpet)
  - 部分的観察できる (Partially observable): 場所、埃の位置 (location, dirt at current location).
  - 視覚(Percept): *[L, Clean]*, i.e.,
  - 始点 #5 か #7

  解?

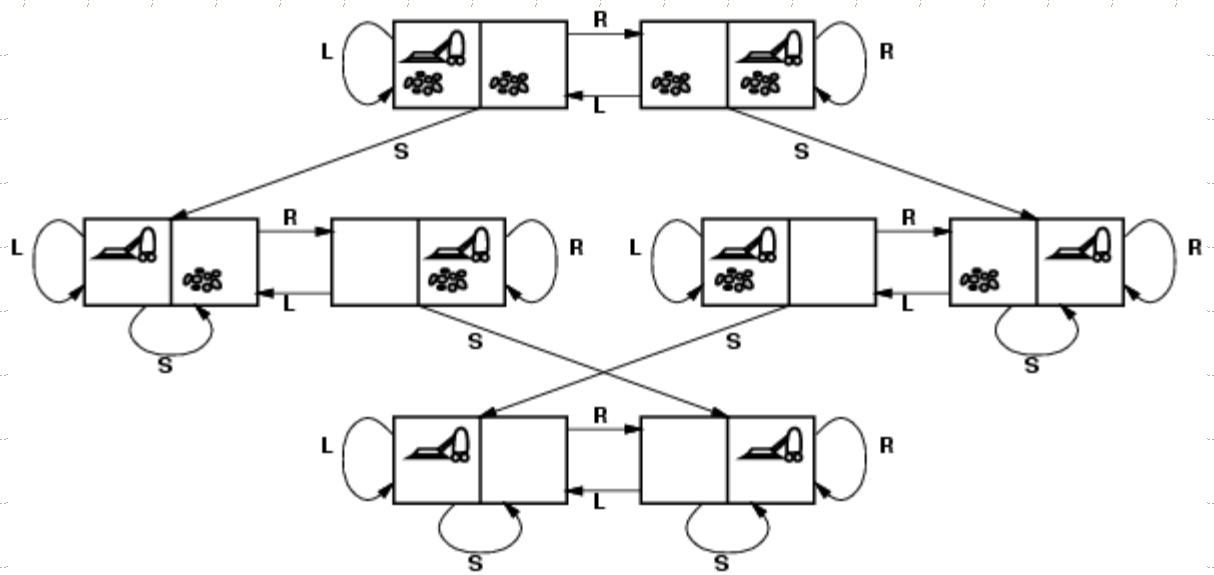
掃除機の世界

- センサない(Sensorless), 始点 {*1,2,3,4,5,6,7,8*} e.g., *Right*（右）なら {*2,4,6,8*}
  解?
  [Right,Suck,Left,Suck]



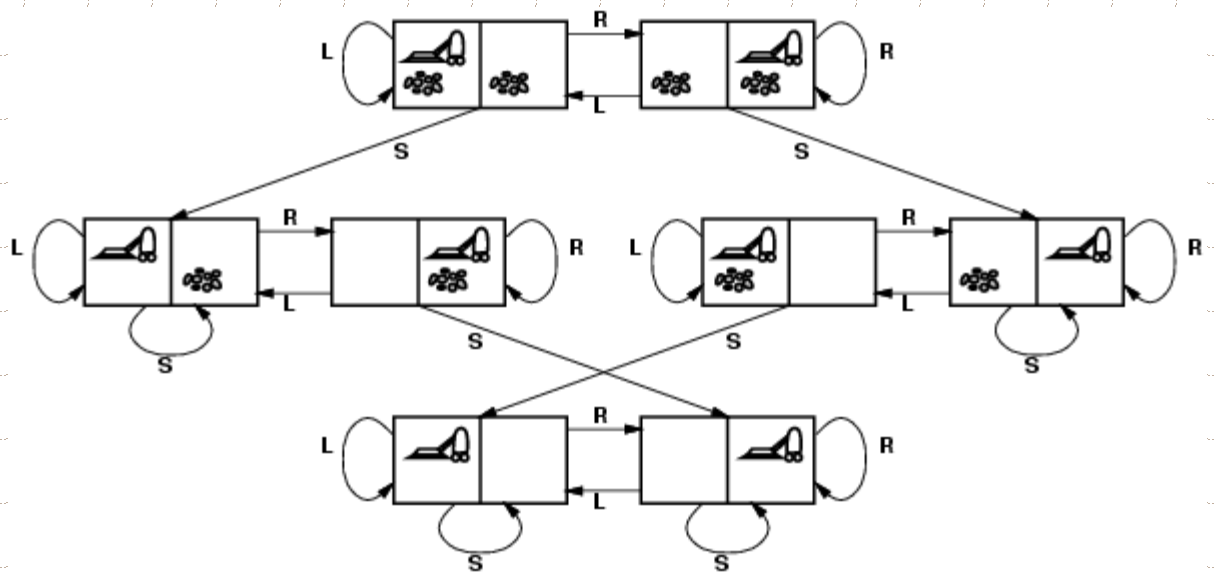
- 偶発的問題(Contingency)
  - 非決定論的(Nondeterministic)
    吸い込みはきれいな部屋に埃を残
    (*Suck* may dirty a clean carpet)
  - 部分的観察できる
    (Partially observable): 場所、埃の位置
    (location, dirt at current location).
  - 視覚(Percept): *[L, Clean]*, 始点 #5 か #7
    解? *[Right, **if** dirt **then** Suck]*

掃除機の世界：状態空間のグラフ (Vacuum world state space)



- 状態(states)?
- 行為(actions)?
- ゴール検査(goal test)?
- 経路コスト(path cost)?

掃除機の世界：状態空間のグラフ (Vacuum world state space)

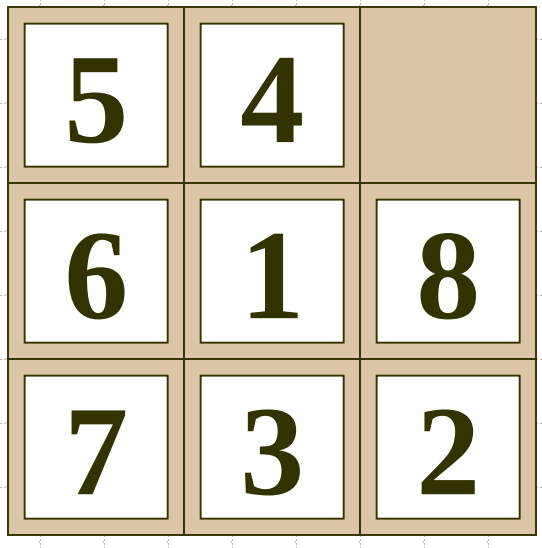


- 状態(states)? 状態1-8の部分集合(integer dirt and robot location)
- 行為(actions)? L:左に動く、R:右に動く、S:吸い込む(*Left*, *Right*, *Suck)*
- ゴール検査(goal test)?埃がない(no dirt at all locations)
- 経路コスト(path cost)? 1各々の行為は、コスト1を要する(1 per action)

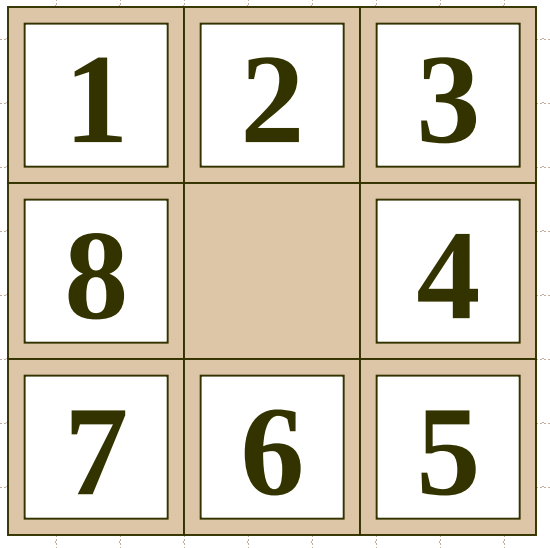
例題1.　8パズル

- 問題：滑りブロックパズルの族に属する。

8つのタイルの各々が9個の区画のどの位置にあるかによって決まる.

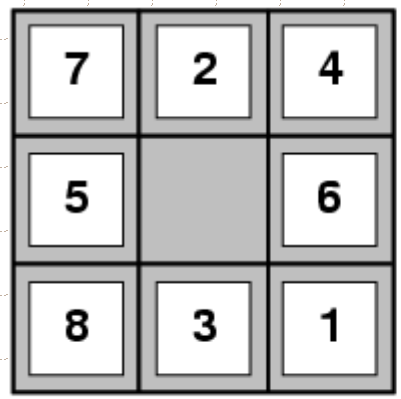


Start State

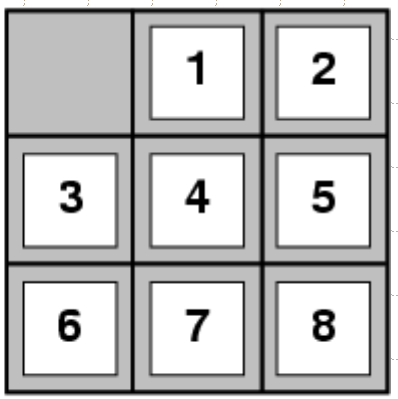


Goal State

例題1.　8パズル



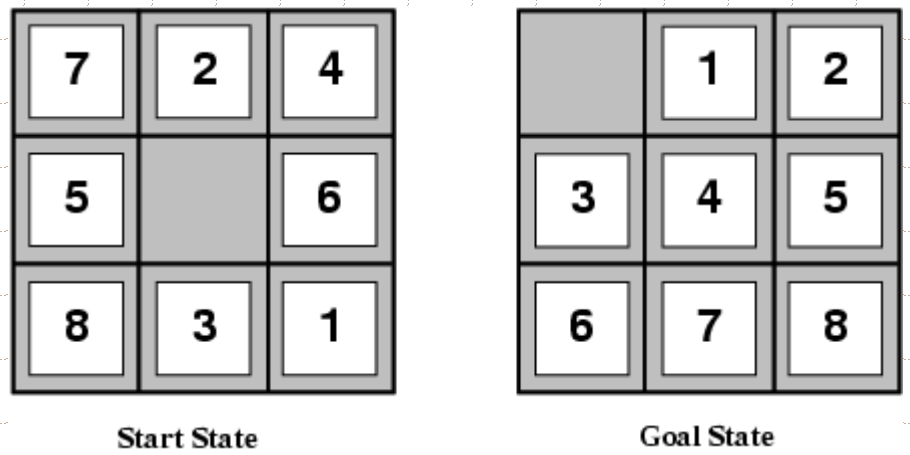
Start State



Goal State

- 状態(states)?
- 行為(actions)?
- ゴール検査(goal test)?
- 経路コスト(path cost)?

例題1.　8パズル



- 状態(states)? タイルの位置(locations of tiles)
- 行為(actions)?空所を上下左右に動かす(move blank left, right, up, down)
- ゴール検査(goal test)?状態上図のゴール配置に一致する
- 経路コスト(path cost)? 各々の行為は、コストは1を要する(1 per move)

[備考: 最適の解のクラスはNP-完全であること(optimal solution of *n*-Puzzle family is NP-hard)]

現実世界の問題

- ルート発見
- 旅行と巡回セールスマン問題
- VLSIレイアウト
- ロボットのナビゲーション
- アセンブリの順序付け

まとめ

- 問題の種類
- 合理的エージェントアプローチ
- タスク環境のPEAS表現
- 問題解決エージェントは，ゴールが与えられて，そのゴールに至る解を探索する．
- はじめに，**ゴール**を定式化し，つぎに，**問題**の定式化を行う．
- 問題は，**初期状態，オペレータ，ゴール検査，経路コスト**から成り立つ．