Agents in the abstract

Objectives:
• Formulate an abstract model for an agent interacting with an environment

• Background reading:
  Chapter 1 of Weiss, MAS, MIT Press, 2000
  Chapter 2 of Wooldridge, Intro MAS. Wiley, 2003
  Chapter 2 of Russel and Norvig, AI, 2nd ed. Prentice-Hall, 2003

Abstract model of an agent

Environment can be characterised as a set of poss. states
\[ S = \{s_1,\ldots,s_k,\ldots\} \]

Agent’s perceptions are another set
\[ P = \{p_1,\ldots,p_n,\ldots\} \]

Each percept can be a tuple of values \((v_1,\ldots,v_s)\)

Its actions are a set:
\[ A = \{a_1,\ldots,a_n,\ldots\} \]

Some action sets may allow actions to be sequences of more primitive actions \([pa_1,pa_2,\ldots,pa_n]\) - including the empty sequence [], or null action

Others may have actions that are tuples \((ca_1,\ldots,ca_m)\) of concurrent actions

Abstract model (cont.)

Agents ability to perceive the environment is represented by a function:

\[ \text{see: } S \rightarrow P \]

Typically see \(s\) is a many-one function since the agent \(s\) sees many different states as the same.

Action selection by agent is a function:

\[ \text{action: } P^* \rightarrow A \]

Where \(P^*\) is the set of all finite sequences of percepts

(Agent will use the function to determine its next action based on a complete or partial history of its perceptions up to current time.)
Agent behaviour

The **behaviour** of an agent $Ag$ with perception and action functions: $see$, $action$

In an environment characterised by a state transition relation: $env$

Is the **set** of interaction histories.

An **interaction history** is a sequence of pairs:

$H: <s_1,a_1>,<s_2,a_2>,...<s_k,a_k>,...$

where, $s_i$ in $S$ and for all $i$: $a_i = action(<see(s_1), see(s_2),..., see(s_i-1)>)$ and $env(a_i, s_i, s_{i+1})$

---

Agent maintenance tasks

Agent’s can have an assigned **maintenance task** represented by a predicate $Maintain$
on environment states

Agent fulfils its maintenance task if every history that begins with a state $s_1$ such that $Maintain(s_1)$ is such that $Maintain(s_i)$ holds for all $i>1$

---

Terminated interactions

Interaction history may be infinite

An interaction history **terminates** at some point $t$,

iff for all $i>t$ $a_i=NULL$

i.e. it has the form:

$<s_1,a_1>,<s_2,a_2>,...<s_i,a_i>, <s_{i+1},null>, <s_{i+2},null>, s_{i+3},null>, ...$

---

Agent achievement tasks

Agent’s can have a specific **achievement task** represented by a pair of predicates $Start$, $Goal$
on environment states

Agent achieves its goal if every history that begins with a state $s_i$ such that $Start(s_i)$ terminates or essentially terminates at a point $t$ such that $Goal(s_t)$

Agent achieves and maintains its goal if every history that begins with a state $s_i$ such that $Start(s_i)$ contains a state $s_t$ such that $Goal(s_t)$ and for all $i>t$ $Goal(s_i)$
**Tropistic agents – simple reflex agents with no memory**

Action function returns an action based on single percept – the last percept. So:

\[
\text{action: } P \to A
\]

**Percept-test → action rules**

Action function of a tropistic agent is typically specified by a set of

\[
\text{percept-test } \to \text{ action rules.}
\]

For a very simple agent could even be a table pairing percepts With actions.

\[
\text{e.g teleo-reactive programs of robotics course}
\]

- \text{ClearAhead } \to \text{ move_forward}
- \text{ObstacleAhead } \to \text{ turn_right}

**Example mine agent collecting bar of gold**

Start Position Facing E

\[
\begin{array}{c}
\text{cell(1,1)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{cell(m,n)} \\
\end{array}
\]

Goal Destination With gold Out of cart

**Tropistic Agent Control Program**

While true do {
    \text{NextActionTime} = \text{now()} + \text{CT};
    \text{do(action(see()))};
    \text{waitUntil(NextActionTime)}
}

\text{CT} is the agent cycle time - it determines how frequently it senses its environment and acts upon it.

(Woolridge agent cycle does not include CT concept in agent program - can be implicit as a delay in the implementation of the see function.)
States and percepts

Truck agent can be in any of the \( m \times n \) cells facing e, w, n or s

Gold can be in any of the cells or in the cart \((m \times n + 1)\) possibilities

**So environment \( S \) has \( 4m \times n(m \times n + 1) \) states.**

Suppose agent has sensors and can detect:
- whether or not there is a wall to the front, left or right of the cart in current cell, which direction it is facing.
- whether or not a bar of gold is in the same cell, whether or not it is loaded with a bar of gold.

Its percepts are:
- \( \text{wallF, wallL, wallR, not(wallF), not(wallL), not(wallR)} \)
- \( \text{facing(C)}, \) where \( C \) is n, s, e, or w
- \( \text{loaded, not(loaded), goldSC, not(goldSC)} \)

Agent actions

It can move to a connected adjacent cell in front of it with a moveF action.

It can also turn 90 degrees to left or right with turn(l), turn(r) actions.

It can put the gold in the cart if it is in same cell as cart with: pickup action.

It can take the gold out with a: putout action.

**We assume actions always succeed - deterministic outcome.**

Action function defined by set of rules applied to a list of percepts returning a list of actions

- \([\text{loaded, facing(s), wallF}] \implies [\text{turn(l)}] \)
- \([\text{loaded, facing(e), wallF, wallR}] \implies [\text{putout}] \)
- \([\text{loaded, facing(s), not(wallF)}] \implies [\text{moveF}] \)
- \([\text{loaded, facing(e), not(wallF)}] \implies [\text{moveF}] \)
- \([\text{not(loaded), facing(e), wallF, wallR}] \implies [] \)
- \([\text{not(loaded), facing(w), wallF, wallL, not(goldSC)}] \implies [] \)
- \([\text{not(loaded), facing(w), goldSC}] \implies [\text{pickup, turn(l)}] \)
- \([\text{not(loaded), facing(e), goldSC}] \implies [\text{pickup, turn(r)}] \)
- \([\text{not(loaded), facing(e), not(wallF), not(goldSC)}] \implies [\text{moveF}] \)
- \([\text{not(loaded), facing(w), not(wallF) not(goldSC)}] \implies [\text{moveF}] \)
- \([\text{not(loaded), facing(e), not(wallR), wallF, not(goldSC)}] \implies [\text{turn(r), moveF, turn(r)}] \)
- \([\text{not(loaded), facing(w), not(wallL), wallF, not(goldSC)}] \implies [\text{turn(l), moveF, turn(l)}] \)
**Hysteretic agents – agents with memory state**

Action function returns an action based on internal state that 'remembers' in some way the sequence of received percepts.

**Functional description of hysteretic agent**

Memory has some set of states $M$

Memory is updated given a new percept with a function:

$$
\text{next}: P \times M \rightarrow M
$$

$M$ records the new percept and (in some way) all the previously received percepts, perhaps not perfectly. Memory may fade.

Action function, which should be based on history of percepts, can use memory state and the new percept:

$$
\text{action}: P \times M \rightarrow A
$$

**Hysteretic Agent Control Program**

Initialise $m$;

while true do {
  NextActionTime=now()+CT;
  p:=see();
  do(action(m,p));
  m:=next(p,m);
  waitUntil(NextActionTime)
}

see accesses current environment state, do acts on this state

Wooldridge has: action: $M \rightarrow A$ and loop body instructions:

$$
p:=\text{see}();
\text{m:=next}(p,m);
\text{do}(\text{action}(m));
$$

Equivalent if $m$ remembers *all* that is relevant in percept $p$

**Combined action and next function - response**

Often convenient to define just one function:

$$
\text{response}: M \times P \rightarrow M \times A
$$

which maps the current memory state and new percept into the new memory state and the action response

When $M$ is finite, this function characterises a finite state transducer with input alphabet $P$ and output alphabet $A$
**Example hysteretic agent**

- Suppose our mine traversing cart agent does not have a direction sensor or a load sensor
- We still assume it starts in cell 1,1 facing east and that its actions always succeed
- How does it traverse the maze?
- It has to *remember* its direction and whether or not it is loaded

**Model of agent with a response function**

**New Agent Control Program**

Initialise $m$;
while true do {
    NextActionTime = now()+CT;
    $(m,a) := \text{response}(m,\text{see}());$
    do(a);
    waitUntil(NextActionTime)}

**Response function**

$$M = \{ \text{facing}(C) | C \in \{e, w, n, s\} \} \times \{ \text{loaded, not(loaded)} \}$$

initial value is $\text{facing}(e), \text{not(loaded)}$

$$(\text{not(loaded)}, \text{facing}(e)), \text{not(\text{wallF}, \text{not(goldSC)}}) \}$$

$$(\text{not(loaded)}, \text{facing}(w)), \text{not(\text{wallF}, \text{not(goldSC)}}) \}$$

$$(\text{not(loaded)}, \text{facing}(e)), \text{not(\text{wallR}, \text{wallF, not(goldSC)}}) \}$$

... $$(\text{not(loaded)}, \text{facing}(w)), \text{not(goldSC)} \}$$

$$(\text{loaded}, \text{facing}(s)), \text{[pickup, turn(l)]} \}$$

... $$(\text{loaded}, \text{facing}(s)), \text{[not(wallF)}] \}$$

... $$(\text{loaded}, \text{facing}(e)), \text{[wallF]} \}$$

... $$(\text{loaded}, \text{facing}(e)), \text{[not(loaded), facing(e)]}, \text{[putout]} \}$$
**Agents with an explicit goal**

- Our program has the cart’s achievement goal *implicit in its rules for determining an action.*

- We could make the goal explicit. Could represent it somehow in internal memory of the cart agent. Makes the agent more flexible.
  
e.g. could have the target cell explicitly represented as a goal

- For different target cells action function will be different, especially when delivering the gold

- Also, goal might be to find and transport to goal cell two bars of gold.

---

**Model of agent with an explicit goal**

**Functional description of agent with an explicitly given goal**

Only difference is agent’s internal state is now $M \times G$ where $G$ is a set of goals

response: $M \times G \times P \rightarrow M \times A$
Control program for hysteretic agent with an explicit goal

Initialise m;
while true do {
    NextActionTime = now() + CT;
    (m, a) := response(m, g, see());
    do(a);
    waitUntil(NextActionTime)}

Example agent with an explicitly given goal

• Imagine our cart agent has a memory component that tells it how many bars, `nbars`, to pick up during its search through the mine

• It is still implicit that it must systematically search and collect bars of gold to be delivered to cell m,n. It is explicit that it should terminate its search when it has collected `nbars`.

• `nbars` is part of agent state.

• The agent counts how many bars it has collected, by updating its memory each time it picks up a bar. It goes straight to delivery cell m,n only when it has collected the required number.

Agent with several goals and a plan library

• If an agent is given a set of goals we might want to provide it with a library of alternative action plans. Each action plan is a response function for a particular goal.

• Agent has one of its goals as the current goal.

• As its memory state and percepts change, it can update its current goal and/or change its response function plan for the current goal.

• This allows the agent to:
  • alternate between goals/tasks
  • abandon infeasible goals
  • abandon infeasible plans

Model of agent with plans

goals
plans
update current goal and plan

see

current goal
memory

environment
MAS05 - Agents in the abstract

January 10, 05 35

Reactive agents

Reactive agents are essentially hysteretic agents, usually with no explicit goals and at most finite internal memory component.

Russel and Norvik (in second edition) call them:

Model based reflex agents

because internal state can model part of the environment

January 10, 05 36

Switching behaviour on key percept or action

- Even without an explicitly represented goal it is still useful to be able to select a response function from a library of response functions

- Switch is usually made on a key percept/memory state configuration - the trigger for the switch

- Occurrence of the trigger leads to adoption of a different implicit goal represented by the new response function

- In production rule systems, change of the set of rules being used to determine next action is called an agenda switch

January 10, 05 37

Control program for agent with several goals and a plan library

Initialise m,g,r;
while true do {
  NextActionTime=now()+CT;
p:= see();
(m,a) := r(m,p);
doa);
(g,r):=selectGoalAndPlan(m,g,p,Goals,Plans);
waitUntil(NextActionTime)}
Trigger switching of response function

Finding Gold Response Function

Switching response functions for cart agent

- Cart starts using Find Gold response function
- Switch trigger is: gold_same_cell percept
- On trigger, agent switches to Deliver Gold response function