Reasoning Agents

José M Vidal

Department of Computer Science and Engineering
University of South Carolina

August 30, 2005

Abstract

Basic notation and frameworks for reasoning agents [Vlassis, 2003, Chapter 2] [Wooldridge, 2002, Chapters 3–4].

Utility Maximizing Agents

- Assume agents inhabit stochastic world defined by a Markov process where
  - s_t is a state
  - a_t is an action
  - $P(s_{t+1} \mid s_t, a_t)$ is the transition function.
- The agent has some **goals** it wants to achieve.
- How do we map these goals into the Markov process?

From Goals to Utilities

- A goal is just a preference over certain states.
- Utility function $U(s)$ is the utility of state s for the agent.
- The agent in s_t should take the action a_t^* which maximizes its expected utility

$$a_t^* = \arg\max_{a_t \in A} \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) U(s_{t+1})$$

- The function that implements this choice is the policy. In this case:

$$\pi^*(s) = \arg\max_{a} \sum_{s'} P(s' \mid s, a) U(s')$$

Greed is Good?

$$a_t^* = \arg\max_{a_t \in A} \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) U(s_{t+1})$$

- Is this greedy policy the best?

Greed is Good?

$$a_t^* = \arg\max_{a_t \in A} \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) U(s_{t+1})$$

- Is this greedy policy the best?
- No. The agent could get stuck in a subset of states that is suboptimal.

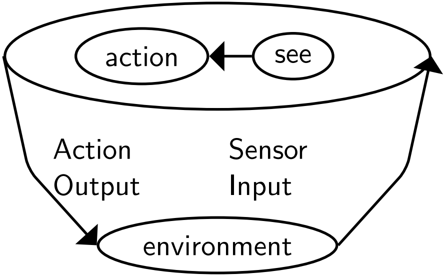
Greed is Good?

$$a_t^* = \arg\max_{a_t \in A} \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) U(s_{t+1})$$

- Is this greedy policy the best?
- No. The agent could get stuck in a subset of states that is suboptimal.
- Instead, discount future utilities by some constant $0 > \gamma < 1$ for each step.
- Optimal policy can be found using reinforcement learning.

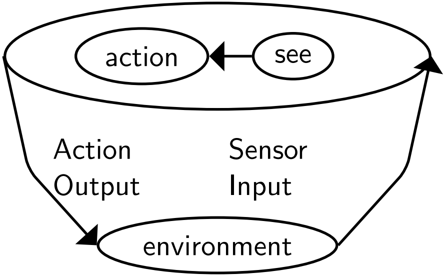
Deductive Reasoning Agents

- The goal is to implement an agent as a theorem-prover.
- The transduction problem is translating from the real world into good symbolic descriptions.
- The reasoning problem is getting agents to manipulate and reason with this knowledge.



Deductive Reasoning Agents

- The goal is to implement an agent as a theorem-prover.
- The transduction problem is translating from the real world into good symbolic descriptions.
- The reasoning problem is getting agents to manipulate and reason with this knowledge.



- *Counterpoint*: Rodney Brooks believes that the world should be its own model—an idea supported by Herbert Simon's example of an ant walking in the sand.

Agents as Theorem Provers

- The agent has a database Δ of statements such as
  ```
  open(valve221)
  dirt(0,1)
  in(3,2)
  dirt(x,y) ∧ in(x,y) → do(clean).
  ```
- The last one is a deduction rule, the set of all of them is p.
- We write $\Delta \rightarrow_p x$ if x can be derived from Δ using p.
- The `see` and `next` functions from the agent with state remain the same. The `action` function has to be redefined.

Action Selection

```
function action(Δ:D) returns an action
begin
  for each a ∈ Actions do
    if Δ →_p Do(a) then
      return a
    end-if
  end-for
  for each a ∈ Actions do
    if ¬(Δ →_p ¬Do(a)) then
      return a
    end-if
  end-for
  return nil
end
```

Vacuum World

Dirt (0,2)	Dirt (1,2)	(2,2)
(0,1)	Robot (1,1)	(2,1)
(0,0)	(1,0)	(2,0)

- `In(x,y)`: agent is at x,y.
- `Dirt(x,y)`: there is dirt at x,y.
- `Facing(d)`: agent is facing direction d.
- Possible updating rules:

Vacuum World

Dirt (0,2)	Dirt (1,2)	(2,2)
(0,1)	Robot (1,1)	(2,1)
(0,0)	(1,0)	(2,0)

- `In(x,y)`: agent is at x,y.
- `Dirt(x,y)`: there is dirt at x,y.
- `Facing(d)`: agent is facing direction d.
- Possible updating rules:

$\texttt{In(x,y)} \wedge \texttt{Dirt(x,y)} \rightarrow \texttt{Do(suck)}$

$\texttt{In(0,0)} \wedge \texttt{Facing(north)} \wedge \neg\texttt{Dirt(0,0)} \rightarrow \texttt{Do(forward)}$

$\texttt{In(0,1)} \wedge \texttt{Facing(north)} \wedge \neg\texttt{Dirt(0,1)} \rightarrow \texttt{Do(forward)}$

$\texttt{In(0,2)} \wedge \texttt{Facing(north)} \wedge \neg\texttt{Dirt(0,2)} \rightarrow \texttt{Do(turn)}$

$\texttt{In(0,2)} \wedge \texttt{Facing(east)} \rightarrow \texttt{Do(forward)}$

Vacuum Exercise

Dirt (0,2)	Dirt (1,2)	(2,2)
(0,1)	Robot (1,1)	(2,1)
(0,0)	(1,0)	(2,0)

- What are the rules for picking up all the dirt, wherever it may be?

Vacuum Exercise

Dirt (0,2)	Dirt (1,2)	(2,2)
(0,1)	Robot (1,1)	(2,1)
(0,0)	(1,0)	(2,0)

- What are the rules for picking up all the dirt, wherever it may be?
- How about:

```
In(x,y) ∧ Dirt(x,y) → Do(suck)
In(x,y) ∧ ¬Dirt(x,y) ∧ ¬ Pebble(x,y) →
      Do(drop-pebble)
In(x,y) ∧ Dirt(a,b) ∧ (a ≠ x ∨ b ≠ y) →
      Do(turn-towards(a,b)) ∧ Do(forward)
```

Pragmatics

- Building a purely logical agent is impractical.

Pragmatics

- Building a purely logical agent is impractical.
- Proving theorems in first-order predicate logic is slow.

Pragmatics

- Building a purely logical agent is impractical.
- Proving theorems in first-order predicate logic is slow.
- Late actions are based on old information.

Pragmatics

- Building a purely logical agent is impractical.
- Proving theorems in first-order predicate logic is slow.
- Late actions are based on old information.
- But, webservice description languages are built to be used by logical agents. There might be a new renaissance of logical approaches.

Pragmatics

- Building a purely logical agent is impractical.
- Proving theorems in first-order predicate logic is slow.
- Late actions are based on old information.
- But, webservice description languages are built to be used by logical agents. There might be a new renaissance of logical approaches.
- For example, OWL-S uses service profiles which define services in terms of their Inputs, Outputs, Pre-conditions, and Effects (IOPEs).

Software For Deductive Reasoning

- Prolog programming language. Uses backtracking.
- Jess language. Uses the Rete algorithm (forward).
- SOAR cognitive architecture. Uses backtracking and chunking.
- Many more automated theorem provers are available.

Agent-Oriented Programming

- Program agents in terms of mentalistic notions. Pre-cursor to a lot of important work in agent research.
- The hope is that using these abstractions would simplify the programming of agents.
- Introduced by Yoav Shoham in 1990.
- The idea was then implemented as AGENT0 [Shoham, 1991]. Not used anymore.

AOP Primitives

- An agent has
  - Capabilities things it can do.
  - Beliefs
  - Commitments things it means to do.
  - Commitment rules that tell it when to create or drop a commitment.
- The commitment rules have a message condition and a mental condition (both in the conditional part).
- An agent can take a private action which amounts to running a subroutine, or a communicative action which amounts to sending a message.
- Messages are limited to: requests, unrequests, and inform messages.

AGENT0 Example

```
COMMIT(
  (agent, REQUEST, DO(time, action)) ;msg condition
  (B,
    [now, Friend agent] AND
    CAN(self, action) AND
    NOT [time, CMT(self, anyact)]) ;metal condition
  self,
  DO(time, action))
```

AGENT0 Example

```
COMMIT(
  (agent, REQUEST, DO(time, action)) ;msg condition
  (B,
    [now, Friend agent] AND
    CAN(self, action) AND
    NOT [time, CMT(self, anyact)]) ;metal condition
  self,
  DO(time, action))
```

- If I receive a message from agent which requests for me to do action at time and I believe that
  - agent is a friend.
  - I can do action
  - at time, I am not committed to doing any other action.
- then commit to doing action at time.

Practical Reasoning Agents

- Practical reasoning is reasoning directed towards action.

Bratman, 1990.

Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes.

- Humans usually divide it into
  - Deliberation determining what state we want to achieve.
  - Means-end reasoning deciding how to achieve this state of affairs.

Constraints

- There is only so much CPU and memory. An agent must deal with these resource bounds by somehow constraining its reasoning.
- The world does not stop for the agent. An agent must perform under certain time constraints.
- How do we control an agent's reasoning?
- How do we know when to stop thinking?

Constraints

- There is only so much CPU and memory. An agent must deal with these resource bounds by somehow constraining its reasoning.
- The world does not stop for the agent. An agent must perform under certain time constraints.
- How do we control an agent's reasoning? (think only about what is important, but, how does it know what is important, it should think about what is important before thinking about stuff, but...).
- How do we know when to stop thinking? (if I only have 5 more minutes I could solve this problem, no, wait, another 5, really.)

Intentions

- Intentions are pro-attitudes; they tend to lead to action.

Intentions

- Intentions are pro-attitudes; they tend to lead to action.
- If you intend to do p then I can assume that you will make a reasonable attempt at achieving p.

Intentions

- Intentions are pro-attitudes; they tend to lead to action.
- If you intend to do p then I can assume that you will make a reasonable attempt at achieving p.
- I can also assume that you will only intend p if you think it is not impossible to achieve.

Intentions

- Intentions are pro-attitudes; they tend to lead to action.
- If you intend to do *p* then I can assume that you will make a reasonable attempt at achieving *p*.
- I can also assume that you will only intend p if you think it is not impossible to achieve.
- The fact that I intend *p* does not mean that I necessarily have to act on it—there might be obstacles in my way.

Intentions

- Intentions are pro-attitudes; they tend to lead to action.
- If you intend to do p then I can assume that you will make a reasonable attempt at achieving p.
- I can also assume that you will only intend p if you think it is not impossible to achieve.
- The fact that I intend p does not mean that I necessarily have to act on it—there might be obstacles in my way.
- I should not persist with my intentions for too long. If they have failed for some time maybe it is time to give up.

Advantages of Intentions

- Drive means-ends reasoning once agent has an intention then it can focus reasoning on achieving that goal.
- persist so agent will not give up at the first sign of trouble.
- constrain future deliberation so agent does not consider options that are inconsistent with current intentions.
- influence beliefs upon which future practical reasoning is based so the agent can plan assuming the intention will be achieved.

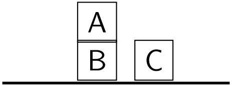
Means-Ends Reasoning

- Means-end reasoning is the process of deciding how to achieve an end using the available means, you will remember this from your AI class as planning.
- A planner is composed of
  - A goal that the agent intends to achieve.
  - The current state of the environment.
  - The actions available to the agents (aka operators).
- The output of a planner is a plan.
- A plan consists of the series of actions that must be taken to get from the current state to the goal state.

Blocks World Example

Blocks World Example

Start



Clear(A),
On(A,B),
OnTable(B),
OnTable(C),
Clear(C)

Goal

Clear(A),
Clear(B),
Clear(C),
OnTable(A),
OnTable(B),
OnTable(C)

Blocks World Example

Stack(x,y)

Start

Goal

Stack(x,y)

precondition: Clear(y), Holding(x)

delete: Clear(y), Holding(x)

add: ArmEmpty, On(x,y)

Blocks World Example

Stack(x,y)

UnStack(x,y)

UnStack(x,y)
precondition: On(x,y), Clear(x), ArmEmpty
delete: On(x,y), ArmEmpty
add: Holding(x), Clear(y);

Blocks World Example

Stack(x,y)

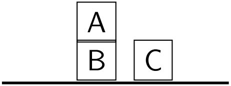
UnStack(x,y)

Pickup(x)

Pickup(x)
precondition: Clear(x), OnTable(x), ArmEmpty
delete: OnTable(x), ArmEmpty
add: Holding(x)

Blocks World Example

Start



Goal

Stack(x,y)

UnStack(x,y)

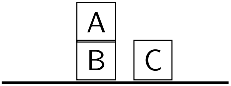
Pickup(x)

PutDown(x)

PutDown(x)
precondition: Holding(x)
delete: Holding(x)
add: ArmEmpty, OnTable(x)

Blocks World Example

Start



Goal

Stack(x,y)

UnStack(x,y)

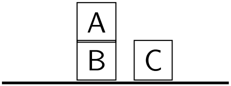
Pickup(x)

PutDown(x)

What predicates should we use?

Blocks World Example

Start



Goal

Stack(x,y)

UnStack(x,y)

Pickup(x)

PutDown(x)

Solution:
UnStack(a,b)
PutDown(a)

Blocks World Example

Stack(x,y)

UnStack(x,y)

Pickup(x)

PutDown(x)

- In general, generating a plan can take exponential time on the set of operators.
- We do not need to generate a whole plan before acting, especially since our intentions might change in the meantime.

Practical Reasoning Agent

```
B = initial-beliefs; I = initial-intentions
while true :
  p = see(perceptions())
  B = belief-revision-function(B,p)
  D = options(B,I) #determine agent's goals
  I = filter(B,D,I) #which goals?
  plan = plan(B,I) #generate the plan
  while not (empty(plan) or done(I,B) or impossible(I,B)):
    next-action = head(plan)
    execute(next-action) #do it.
    plan = tail(plan)
    B = belief-revision-function(B,see(perceptions()))
    if reconsider(I,B): #should agent drop goals?
      D = options(B,I)
      I = filter(B,D,I)
    if not sound(plan, I, B): #the plan might not be good.
      plan = plan(B,I)
```

Commitment

- There are many types of commitment.
- Blind commitment (fanatical): agent continues to maintain intention until it has been achieved.
- Single-minded commitment: agent will continue to maintain intention until it has been achieved or it is impossible to achieve.
- Open-minded commitment: agent will maintain intention as long at it believes it is possible.

Reconsider Commitment

- When to reconsider intentions?

Reconsider Commitment

- When to reconsider intentions?
- If too much then agent spends all the time re-considering, if too little then might end up doing the wrong thing.
- If the world does not change much then agents that stick with the same plan do better.
- If the world changes a lot then agents that re-consider their intentions do better.
- Why?

Procedural Reasoning Systems

- BDI ideas have been implemented in several research and commercial products:
  - The SRI PRS system.
  - The JAM agent.
  - Click here for a list of current agent architectures.

References I

Shoham, Y. (1991).
AGENT0: A simple agent language and its interpreter.
In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 704–709. AAAI Press, Menlo Park, CA.

Vlassis, N. (2003).
A concise introduction to multiagent systems and distributed AI.
Informatics Institute, University of Amsterdam.
http://www.science.uva.nl/~vlassis/cimasdai.

Wooldridge, M. (2002).
Introduction to MultiAgent Systems.
John Wiley and Sons.