Deductive Reasoning Agents
Based on “An Introduction to MultiAgent Systems” and slides by Michael Wooldridge
Agent Architectures

- An agent architecture is a software design for an agent
- Top-level decomposition into perception – state – decision – action
- An agent architecture defines:
  - key data structures
  - operations on data structures
  - control flow between operations
Types of Agents

- 1950’s – now: *Symbolic Reasoning Agents*
  Its purest expression, proposes that agents use explicit logical reasoning in order to decide what to do

- 1980’s – now: *Reactive Agents*
  Problems with symbolic reasoning led to a reaction against this — led to the reactive agents movement

- 1990’s – now: *Hybrid Agents*
  Hybrid architectures attempt to combine the best of symbolic and reactive architectures
Deductive Reasoning Agents

- Traditional approach to build AI systems (symbolic AI)
  - Symbolic representation of environment and behavior
  - Syntactic manipulation of symbolic representation
- Symbolic representation $\rightarrow$ logical formulae
- Syntactic manipulation $\rightarrow$ logical deduction (theorem proving)
Key Problems

state: 
Dist(me, d1)=5, Door(d1)

plan: 
stop!

action: 
brake!

Room 2.21
The problem of translating the real world into an accurate, adequate symbolic description, in time for that description to be useful.

... vision, speech understanding, learning
The problem of how to symbolically represent information about complex real-world entities and processes, and how to get agents to reason with this information in time for the results to be useful.

...knowledge representation, automated reasoning, automatic planning.
Agents as Theorem Provers

- Theory of agency $\varphi$ – some theory that explains how an intelligent agent should behave to optimize some performance measure
- Theory $\varphi$ is viewed as executable specification that is directly executed in order to produce the agent’s behavior
Agents as Theorem Provers

Deliberate agents – simple model of logic-based agents

- Internal state assumed to be a database of formulae (predicate logic)

Example

Open(valve221)
Temperature(reactor4726, 321)
Pressure(tank776, 28)

- Analogous to beliefs in humans – internal state may include incorrect (outdated, invalid) information
Agents as Theorem Provers

Let:

- $L$ be the set of formulae of classical first-order logic
- $D = 2^L$ be the set of $L$ databases
  - $DB, DB_1, \ldots$ are the members of $D$
  - $DB$ represents the internal state of an agent
- $\rho$ is a set of deduction rules that models the agent’s decision making process
- $DB \vdash_\rho \varphi$ means that the formula $\varphi$ can be proved from the database $DB$ using only the deduction rules $\rho$
Agents with State

- Perception function *see*
- Next state function *next*
- Action-selection function *action*
Agents as Theorem Provers

- Perception function

\[ \text{see} : E \rightarrow \text{Per} \]

- Next function

\[ \text{next} : D \times \text{Per} \rightarrow D \]

- Action-selection function (defined in terms of the agent’s deduction rules)

\[ \text{action} : D \rightarrow \text{Ac} \]
foreach $\alpha \in Ac$ do
  
  if $DB \vdash_\rho Do(\alpha)$ then
    return $\alpha$;
  
  end

end

foreach $\alpha \in Ac$ do
  
  if $DB \not\vdash_\rho \neg Do(\alpha)$ then
    return $\alpha$;
  
  end

end
A small robotic agent that cleans up a room divided into a grid of equally sized squares

Agent is equipped with a dirt sensor and a vacuum cleaner

Agent always has a definite orientation (north, south, east, west)

Agent can move forward one step and turn right $90^\circ$

For simplicity we assume the room is $3 \times 3$ and agent always starts in square 0,0 facing north
possible percepts
Per = \{dirt, null\}

domain predicates describing internal state
ln(x, y) – agent is at (x, y)
Dirt(x, y) – there is dirt at (x, y)
Facing(d) – agent is facing direction d

possible actions
Act = \{forward, suck, turn\}
Let \( old(DB) \) denote a set of old information in \( DB \) that should be removed by function \( next \)

\[
old(DB) = \{ P(t_1, \ldots, t_n) | P \in \{ In, Dirt, Facing \} \land P(t_1, \ldots, t_n) \in DB \}
\]

Introduce \( new \) function that gives a set of new perdicates to add to \( DB \)

\[
new : D \times Per \rightarrow D
\]

Given \( old \) and \( new \) functions, the \( next \) function is defined as

\[
next(DB, p) = (DB \setminus old(DB)) \cup new(DB, p)
\]
Reasoning in Vacuum World

Deduction rules

1. $\text{In}(x, y) \land \text{Dirt}(x, y) \rightarrow \text{Do}(\text{suck})$
2. $\text{In}(0,0) \land \text{Facing}(\text{north}) \land \neg \text{Dirt}(0,0) \rightarrow \text{Do}(\text{forward})$
3. $\text{In}(0,1) \land \text{Facing}(\text{north}) \land \neg \text{Dirt}(0,1) \rightarrow \text{Do}(\text{forward})$
4. $\text{In}(0,2) \land \text{Facing}(\text{north}) \land \neg \text{Dirt}(0,2) \rightarrow \text{Do}(\text{turn})$
5. $\text{In}(0,2) \land \text{Facing}(\text{east}) \land \neg \text{Dirt}(0,2) \rightarrow \text{Do}(\text{forward})$
6. $\cdots$
Problems with Deductive Reasoning

- How to convert video camera input to \{dirt, null\} and how to represent properties of dynamic environment
- Decision making assumes a static environment: \emph{calculative rationality}
  - Decision making process suggests an action that was optimal when the process started
  - If decision making is immediate, then we can discard this problem
- Decision making via theorem proving is complex (it may never terminate)
Agent-oriented programming (AOP) – “new programming paradigm, based on a societal view of computation”

Key idea is directly programming agents in terms of intentional notions (→ intentional stance) like belief, commitment, and intention

AGENT0 is the first (experimental) implementation of AOP
AGENT0 Components

Each agent in AGENT0 has 4 components

1. a set of capabilities (things the agent can do)
2. a set of initial beliefs
3. a set of initial commitments (things the agent will do)
4. a set of commitment rules
AGENT0 Commitment Rules

- The key component, which determines how the agent acts, is the commitment rule set.
- Each commitment rule contains:
  - a message condition
  - a mental condition
  - an action
AGENT0 Decision Cycle

On each decision cycle

- The message condition is matched against the messages the agent has received
- The mental condition is matched against the beliefs of the agent
- If the rule fires, then the agent becomes committed to the action (the action gets added to the agents commitment set)
Communication in AGENT0

- Actions may be
  - *private*: an internally executed computation
  - *communicative*: sending messages

- Messages are constrained to be one of three types
  - *requests* to commit to action
  - *unrequests* to refrain from actions
  - *informs* which pass on information
Commitment Rule

\[
\text{_COMMIT}(\text{agent}, \\
\quad \text{REQUEST, DO(time, action)} \\
\text{), ;;; msg condition} \\
\text{B,} \\
\quad [\text{now, Friend agent}] \text{ AND} \\
\quad \text{CAN(self, action) AND} \\
\quad \text{NOT [time, CMT(self, anyaction)]} \\
\text{), ;;; mental condition} \\
\text{self,} \\
\text{DO(time, action)} 
\)
Commitment Rule

If I receive a message from agent which requests me to do action at time, and I believe that:

- Agent is currently a friend
- I can do the action
- At time, I am not committed to doing any other action

then commit to doing action at time.
AGENT0 Execution Cycle

1. Read all current messages, updating beliefs – and commitments – when necessary
2. Execute all commitments for the current cycle where the capability condition of the associated action is satisfied
3. Goto 1
Concurrent MetateM

- Concurrent MetateM – a multi-agent language based on the direct execution of logical formulae.
- A MetateM system contains a number of concurrently executing agents that can communicate via asynchronous message passing.
- Each agent is programmed by giving it a temporal logical specification of the behavior it should demonstrate.
- Temporal logic is classical logic augmented by modal operators for describing how the truth of propositions changes over time.
Temporal Logic

- □ important (holidays)
  means “it is now, and will always be true that holidays are important”

- ◇ important (holidays)
  means “sometime in the future, holidays will be important”

- ◯ important (holidays)
  means “yesterday (in the previous state), holidays were important”

- (¬friends (us) U apologize (you))
  means “we are not friends until you apologize”

- ◯ apologize (you)
  means “tomorrow (in the next state), you apologize”
MetateM Program

- MetateM program is a collection of rules
  \[ \text{past} \rightarrow \text{future} \]
- Execution proceeds by a process of continually matching rules against a “history”, and firing those rules whose antecedents are satisfied
- The instantiated future-time consequents become \textit{commitments} which must subsequently be satisfied
A Concurrent MetateM system contains a number of agents (objects), each object has 3 attributes

- a name
- an interface (for communication)
  - messages the agent will accept
  - messages the agent may send
- a MetateM program

Example: stack agent

\[\text{stack \ (pop, push) \ [popped, stackfull]}\]
MetateM Execution Cycle

1. Update the history of the agent by receiving messages from other agents and adding them to its history

2. Check which rules fire, by comparing past-time antecedents of each rule against the current history to see which are satisfied

3. Jointly execute the fired rules together with any commitments carried over from previous cycles (several alternative actions may be possible, unsatisfied commitments are carried over to the next cycle)

4. Goto 1
A simple system with 3 agents:

- **rp** – a resource producer
  - it can ‘give’ to only one agent at a time
  - it will commit to give to any agent that asks

- **rc1, rc2** – resource consumers (more and less greedy)
MetateM Example

- \( rp(ask_1, ask_2) [give_1, give_2] : \)
  - \( start \rightarrow □ \lnot (give_1 \land give_2) \)
  - \( ⊙ ask_1 \rightarrow □ give_1 \)
  - \( ⊙ ask_2 \rightarrow □ give_2 \)

- \( rc_1(give_1)[ask_1] : \)
  - \( start \rightarrow ask_1 \)
  - \( ⊙ ask_1 \rightarrow ask_1 \)

- \( rc_2(ask_1, give_2)[ask_2] : \)
  - \( ⊙ (ask_1 \land \lnot ask_2) \rightarrow ask_2 \)