LECTURE 3: DEDUCTIVE AGENTS, REACTIVE AGENTS AND HYBRID AGENTS

An Introduction to Multiagent Systems

CIS 7412, Fall 2011

Agent Architectures

- Pattie Maes (1991):

  '[A] particular methodology for building [agents]. It specifies how ... the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions ... and future internal state of the agent. An architecture encompasses techniques and algorithms that support this methodology.'

- Leslie Kaelbling (1991):

  '[A] specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules. A more abstract view of an architecture is as a general methodology for designing particular modular decompositions for particular tasks.'

Types of Agents

- 1956–present: *Symbolic Reasoning Agents*

  Agents make decisions about what to do via *symbol manipulation*.

  Its purest expression, proposes that agents use *explicit logical reasoning* in order to decide what to do.

- 1985–present: *Reactive Agents*

  Problems with symbolic reasoning led to a reaction against this — led to the *reactive agents* movement, 1985–present.

- 1990-present: *Hybrid Agents*

  Hybrid architectures attempt to combine the best of reasoning and reactive architectures.

Symbolic Reasoning Agents

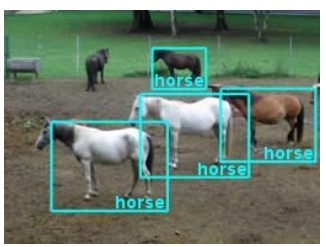
- The classical approach to building agents is to view them as a particular type of knowledge-based system, and bring all the associated methodologies of such systems to bear.
- This paradigm is known as *symbolic AI*.
- We define a deliberative agent or agent architecture to be one that:
  - contains an explicitly represented, symbolic model of the world;
  - makes decisions (for example about what actions to perform) via symbolic reasoning.

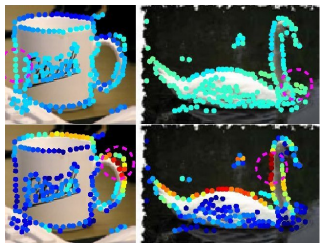
Two Issues

- *The transduction problem*:
  that of translating the real world into an accurate, adequate symbolic description, in time for that description to be useful.
  . . . vision, speech understanding, learning.
- *The representation/reasoning problem*:
  that 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.
- Most researchers accept that neither problem is anywhere near solved.

The transduction problem

- Identifying objects is hard.





The representation/reasoning problem

- Underlying problem lies with the complexity of symbol manipulation algorithms.
- In general many (most) search-based symbol manipulation algorithms of interest are *highly intractable*.
- Hard to find *compact* representations.
- Because of these problems, some researchers have looked to alternative techniques for building agents; we look at these later.

Deductive Reasoning Agents

- How can an agent decide what to do using theorem proving?
- Basic idea is to use logic to encode a theory stating the *best* action to perform in any given situation.
- Let:
  - ρ be this theory (typically a set of rules);
  - Δ be a logical database that describes the current state of the world;
  - Ac be the set of actions the agent can perform;
  - $\Delta \vdash_\rho \phi$ mean that ϕ can be proved from Δ using ρ.

- How does this fit into the abstract description we talked about last time?
- The perception function is as before:

$$see : E \rightarrow Per$$

of course, this is (much) easier said than done.

- The next state function revises the database Δ:

$$next : \Delta \times Per \rightarrow \Delta$$

- And the action function, well a possible action function is on the next slide.

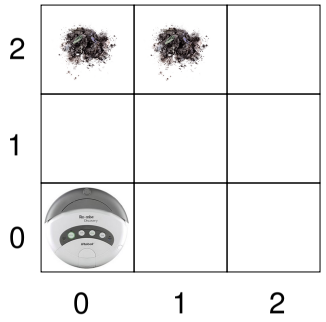

```
for each α ∈ Ac do          /* try to find an action explicitly prescribed */
    if Δ ⊢ρ Do(α) then
        return α
    end-if
end-for

for each α ∈ Ac do          /* try to find an action not excluded */
    if Δ ⊬ρ ¬Do(α) then
        return α
    end-if
end-for

return null                 /* no action found */
```


The Vacuum World

- Goal is for the robot to clear up all dirt.



- Use 3 *domain predicates* in this exercise:

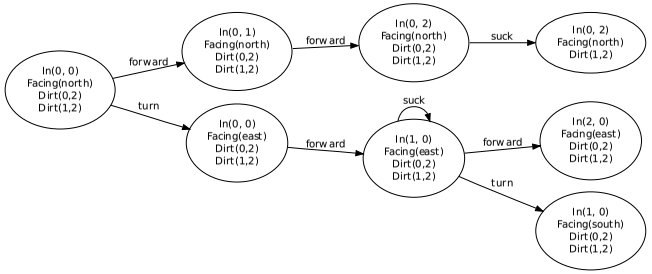
$In(x, y)$	agent is at (x, y)
$Dirt(x, y)$	there is dirt at (x, y)
$Facing(d)$	the agent is facing direction d

- Possible actions:

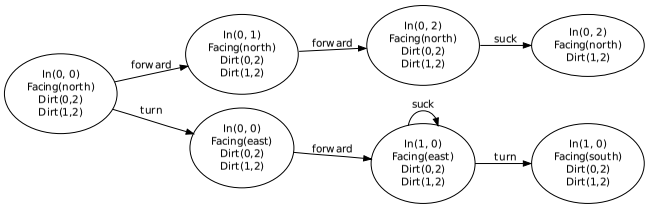
$$Ac = \{turn, forward, suck\}$$

NB: $turn$ means “turn right”.

- With the system as depicted above, here are some possible ways that the system might run.



- If we only write down the distinct states, we get part of the statespace for the problem.



- Rules ρ for determining what to do:

$$\begin{aligned} In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) &\longrightarrow Do(forward) \\ In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) &\longrightarrow Do(forward) \\ In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) &\longrightarrow Do(turn) \\ In(0,2) \wedge Facing(east) &\longrightarrow Do(forward) \end{aligned}$$

- . . . and so on!
- Using these rules (+ other obvious ones), starting at $(0,0)$ the robot will clear up dirt.

- Problems:
  - how to convert video camera input to $Dirt(0,1)$?
  - decision making assumes a *static* environment: *calculative* rationality.
  - decision making using first-order logic is *undecidable*!
- Even where we use *propositional* logic, decision making in the worst case means solving co-NP-complete problems.
  (NB: co-NP-complete = bad news!)
- Typical solutions:
  - weaken the logic;
  - use symbolic, non-logical representations;
  - shift the emphasis of reasoning from *run time* to *design time*.
- We now look at some examples of these approaches.

Agent-oriented programming

- Yoav Shoham introduced "agent-oriented programming" in 1990:

  "new programming paradigm, based on a societal view of computation".

- The key idea:

  directly programming agents in terms of intentional notions like belief, commitment, and intention.

Agent0

- AGENT0 is implemented as an extension to LISP.

  Each agent in AGENT0 has 4 components:

  – a set of capabilities (things the agent can do);
  – a set of initial beliefs;
  – a set of initial commitments (things the agent will do); and
  – a set of *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*; and
  – an action.

- 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).

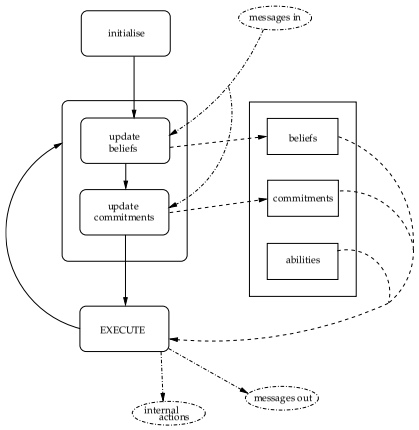
- A commitment rule:

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

- This rule may be paraphrased as follows:
  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*.

- Actions may be
  - *private*:
    an internally executed computation, or
  - *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.



PLACA

- A more refined implementation was developed by Thomas, for her 1993 doctoral thesis.
- Her Planning Communicating Agents (PLACA) language was intended to address one severe drawback to AGENT0: the inability of agents to plan, and communicate requests for action via high-level goals.
- Agents in PLACA are programmed in much the same way as in AGENT0, in terms of *mental change* rules.

- An example mental change rule:

```
(((self ?agent REQUEST (?t (xeroxed ?x)))
 (AND (CAN-ACHIEVE (?t xeroxed ?x)))
      (NOT (BEL (*now* shelving)))
      (NOT (BEL (*now* (vip ?agent))))
 ((ADOPT (INTEND (5pm (xeroxed ?x)))))
 ((?agent self INFORM
       (*now* (INTEND (5pm (xeroxed ?x))))))))
```


- Paraphrased:

  if someone asks you to xerox something, and you can, and you don't believe that they're a VIP, or that you're supposed to be shelving books, then

  – Adopt the intention to xerox it by 5pm, and
  – Inform them of your newly adopted intention.

Concurrent METATEM

- Concurrent METATEM is a multi-agent language in which each agent is programmed by giving it a *temporal logic* specification of the behaviour it should exhibit.
- (Note, though, that the behavior it is capturing is reactive.)
- These specifications are executed directly in order to generate the behaviour of the agent.
- Temporal logic is classical logic augmented by *modal operators* for describing how the truth of propositions changes over time.
- Think of the world as being a number of discrete states.
- There is a single past history, but a number of possible futures — all the possible ways that the world might develop.

- For example. . .

  $\Box$ important(agents)

  means "it is now, and will always be true that agents are important"

  $\Diamond$ important(ConcurrentMetateM)

  means "sometime in the future, ConcurrentMetateM will be important"

  $\blacklozenge$ important(Prolog)

  means "sometime in the past it was true that Prolog was important"

- More examples...

$$(\neg \text{friends(us)}) \,\mathcal{U}\, \text{apologise(you)}$$

means "we are not friends until you apologise"

$$\bigcirc \text{apologise(you)}$$

means "tomorrow (in the next state), you apologise".

- And a last couple of examples ...

$$\circledcirc \text{apologise(you)} \Rightarrow \bigcirc \text{friends(us)}$$

means "if you apologised yesterday, then tomorrow we will be friends".

The operator $\circledcirc$ indicates the previous state in time.

$$\text{friends(us)} \,\mathcal{S}\, \text{apologise(you)}$$

means "we have been friends since you apologised"

- The root of the MetateM concept is Gabbay's *separation theorem*:

  Any arbitrary temporal logic formula can be rewritten in a logically equivalent *past* $\Rightarrow$ *future* form.
- Just like our last example.
- MetateM program is a collection of

$$\textit{past} \Rightarrow \textit{future}$$

rules.
- 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 *commitments* which must subsequently be satisfied.

- An example MetateM program: the resource controller...

$$\circledcirc \text{ask(x)} \Rightarrow \Diamond \text{give(x)}$$
$$\text{give(x)} \wedge \text{give(y)} \Rightarrow \text{(x=y)}$$

  – First rule ensure that an 'ask' is eventually followed by a 'give'.
  – Second rule ensures that only one 'give' is ever performed at any one time.

- A Concurrent MetateM system contains a number of agents (objects), each object has 3 attributes:
  - a name;
  - an interface;
  - a MetateM program.

- An agent's interface contains two sets:
  - messages the agent will *accept*;
  - messages the agent may *send*.
- For example, a 'stack' object's interface:

  stack(pop, push)[popped, stackfull]

  {pop, push} = messages received
  {popped, stackfull} = messages sent

Snow White & The Dwarves

- To illustrate the language Concurrent MetateM in more detail, here are some example programs...
- These are taken from a paper by Michael Fisher.
- Snow White has some sweets (resources), which she will give to the Dwarves (resource consumers).
- She will only give to one dwarf at a time.
- She will always eventually give to a dwarf that asks.

- Here is Snow White, written in Concurrent MetateM:

Snow-White(ask)[give]:

$\bigcirc$ ask(x) $\Rightarrow$ $\Diamond$ give(x)

give(x) $\wedge$ give(y) $\Rightarrow$ (x = y)

- The dwarf ‘eager’ asks for a sweet initially, and then whenever he has just received one, asks again.

eager(give)[ask]:

$$\text{start} \Rightarrow \text{ask(eager)}$$
$$\bigcirc\!\!\!\!\bigcirc\ \text{give(eager)} \Rightarrow \text{ask(eager)}$$

- Some dwarves are even less polite: ‘greedy’ just asks every time.

greedy(give)[ask]:

$$\text{start} \Rightarrow \Box\ \text{ask(greedy)}$$

- Fortunately, some have better manners; ‘courteous’ only asks when ‘eager’ and ‘greedy’ have eaten.

courteous(give)[ask]:

$$((\neg \text{ask(courteous)}\ \mathcal{S}\ \text{give(eager)}) \wedge$$
$$(\neg \text{ask(courteous)}\ \mathcal{S}\ \text{give(greedy)})) \Rightarrow$$
$$\text{ask(courteous)}$$

- And finally, ‘shy’ will only ask for a sweet when no-one else has just asked.

shy(give)[ask]:

$$\textit{start} \Rightarrow \Diamond\ \text{ask(shy)}$$
$$\bigcirc\!\!\!\!\bigcirc\ \text{ask(x)} \Rightarrow \neg\ \text{ask(shy)}$$
$$\bigcirc\!\!\!\!\bigcirc\ \text{give(shy)} \Rightarrow \Diamond\ \text{ask(shy)}$$

- What will happen to ’shy’ when he exists along with greedy?

Reactive Architectures

- There are many unsolved (some would say insoluble) problems associated with symbolic AI.
- These problems have led some researchers to question the viability of the whole paradigm, and to the development of *reactive* architectures.
- Although united by a belief that the assumptions underpinning mainstream AI are in some sense wrong, reactive agent researchers use many different techniques.
- In this presentation, we start by reviewing the work of one of the most vocal critics of mainstream AI: Rodney Brooks.

Brooks — behaviour languages

- Brooks has put forward three theses:
  1. Intelligent behaviour can be generated *without* explicit representations of the kind that symbolic AI proposes.
  2. Intelligent behaviour can be generated *without* explicit abstract reasoning of the kind that symbolic AI proposes.
  3. Intelligence is an *emergent* property of certain complex systems.

- He identifies two key ideas that have informed his research:
  1. Situatedness and embodiment: 'Real' intelligence is situated in the world, not in disembodied systems such as theorem provers or expert systems.
  2. Intelligence and emergence: 'Intelligent' behaviour arises as a result of an agent's interaction with its environment. Also, intelligence is 'in the eye of the beholder'; it is not an innate, isolated property.
- To illustrate his ideas, Brooks built some agents based on his *subsumption architecture*.

- Genghis:

- A subsumption architecture is a hierarchy of task-accomplishing *behaviours*.
- Each behaviour is a rather simple rule-like structure.
- Each behaviour 'competes' with others to exercise control over the agent.
- Lower layers represent more primitive kinds of behaviour, (such as avoiding obstacles), and have precedence over layers further up the hierarchy.

- The resulting systems are, in terms of the amount of computation they do, *extremely* simple.
- Some of the robots do tasks that would be impressive if they were accomplished by symbolic AI systems.
- Steels' Mars explorer system, using the subsumption architecture, achieves near-optimal cooperative performance in simulated 'rock gathering on Mars' domain:

  The objective is to explore a distant planet, and in particular, to collect sample of a precious rock. The location of the samples is not known in advance, but it is *known that they tend to be clustered.*

- For individual (non-cooperative) agents, the lowest-level behavior, (and hence the behavior with the highest "priority") is obstacle avoidance:

  if detect an obstacle *then* change direction.

- Any samples carried by agents are dropped back at the mother-ship:

  if carrying samples *and* at the base *then* drop samples

  if carrying samples and *not* at the base *then* travel up gradient.

  The "gradient" in this case refers to a virtual "hill" that slopes up to the mother ship/base.

- Agents will collect samples they find:

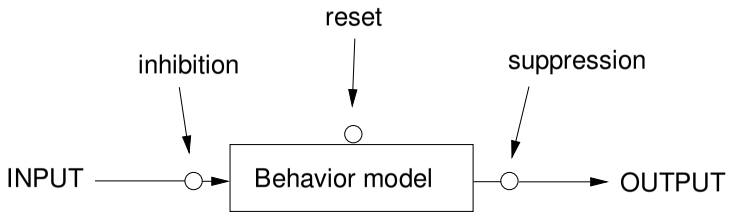
  if detect a sample *then* pick sample up.

- An agent with "nothing better to do" will explore randomly:

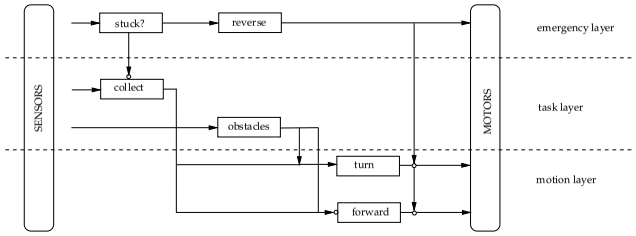
  if true *then* move randomly.

Abstract view of subsumption architecture

- Layered approach based on levels of competence
- Augmented finite state machine:



- A subsumption architecture machine:



- Can build sophisticated machines this way.
- Matarić's Toto was able to map spaces and execute plans all without a symbolic representation.

Situated Automata

- Approach proposed by Rosenschein and Kaelbling.
- An agent is specified in a rule-like (declarative) language.
- Then compiled down to a digital machine, which satisfies the declarative specification.
- This digital machine can operate in a *provable time bound*.
- Reasoning is done *off line*, at *compile time*, rather than *online* at *run time*.

- The theoretical limitations of the approach are not well understood.
- Compilation (with propositional specifications) is equivalent to an NP-complete problem.
- The more expressive the agent specification language, the harder it is to compile it.
- (There are some deep theoretical results which say that after a certain expressiveness, the compilation simply can't be done.)

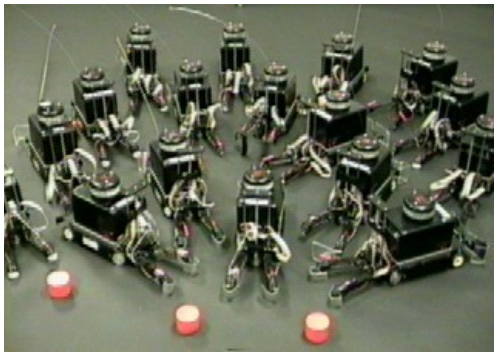
Emergent behaviour

- Important but not well-understood phenomenon
- Often found in behaviour-based/reactive systems
- Agent behaviours "emerge" from interactions of rules with environment.
- Sum is greater than the parts.
  - The interaction links rules in ways that weren't anticipated.

- Coded behaviour
  - In the programming scheme
- Observed behaviour
  - In the eyes of the observer
- There is no one-to-one mapping between the two!
- When observed behaviour "exceeds" programmed behaviour, then we have emergence.

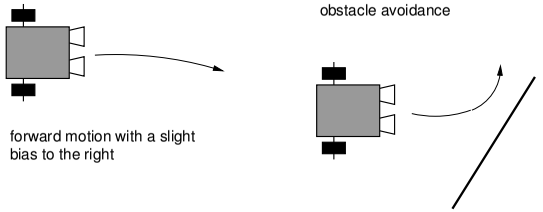
- Emergent flocking.
- Classic example of emergence
  - Reynolds "Boids"
- Program multiple agents:
  - Don't run into any other robot
  - Don't get too far from other robots
  - Keep moving if you can
- When run in parallel on many agents, the result is flocking

- Matarić's "nerd herd" showed flocking behavior:



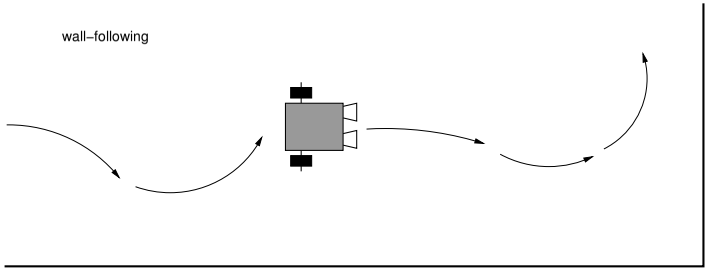
Wall following

- A nice example of emergent behavior.
- We code a robot/agent with two simple behaviors:



- Nothing in there about walls.

- But what we get looks like it was designed to track round the edge of a room.



- It can even find it sway to the middle of the right maze.

- Can also be implemented with these rules:
  - If too far, move closer
  - If too close, move away
  - Otherwise, keep on
- Over time, in an environment with walls, this will result in wall-following
- Is this emergent behavior?

- Can argued yes because
  - Robot itself is not aware of a wall, it only reacts to distance readings
  - Concepts of “wall” and “following” are not stored in the robot’s controller
  - The system is just a collection of rules
- But once I have seen this work, I can program the robot expecting it to happen!

Learning reactive behavior

- We can discover reactive behavior.
- If we have utilities of states, and actions that take our agent from state to state (sound familiar) we can discover the utility of every state.
- The utility of a state e_i is a function of the utility of the states the agent can get to from it (e_j) and the cost of getting to those states:

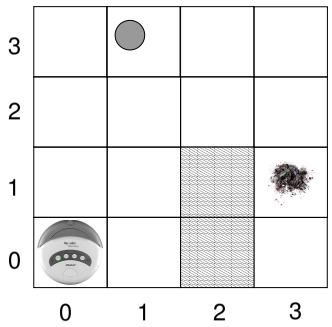
$$V(e_i) = max_j(V(e_j) - c(e_j, e_i))$$

- If we start by assuming all states with unknown utilities have utilities of zero, and recursively update using:

$$V(e_i)_{t+1} = V(e_i)_t + max_j(V(e_j)_t - c(e_j, e_i))$$

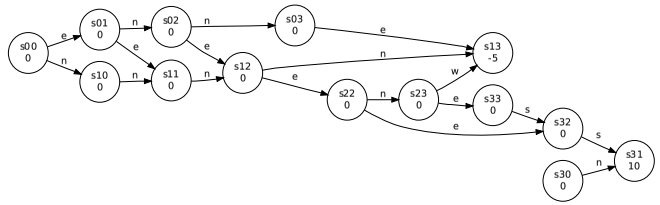
- We can establish the utility of every state.

- Here’s an example world, like Vacuum world, but with some places the robot can’t go, and a place with negative utility (a hole, say).



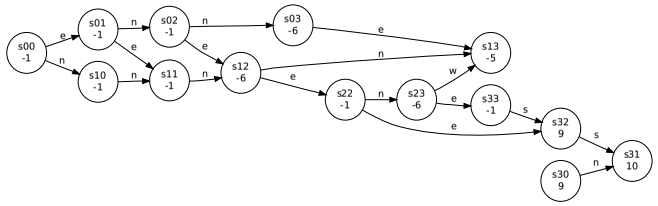
- The robot can go north (up the page), south, east and west. Each action costs 1

- Here is a state/action graph that covers all of the states and some of the actions and initial utilities.

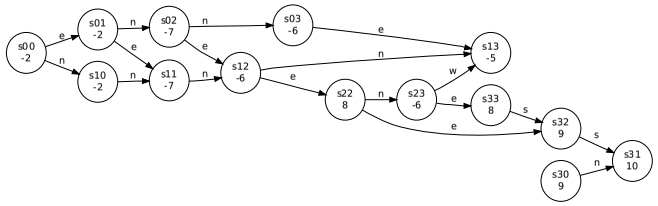


- Now let’s run the the recursive updates.

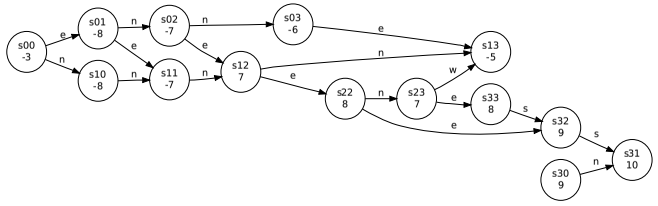
- After one step.



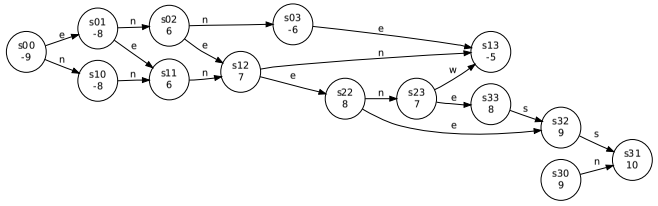
- After two steps.



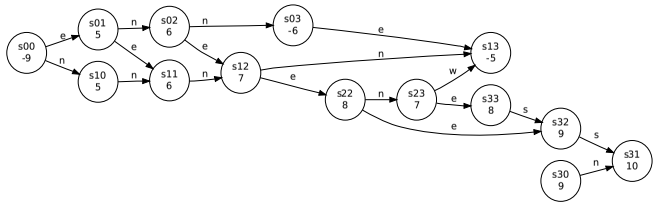
- After three steps.



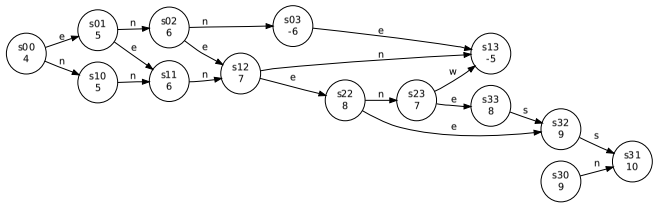
- After four steps.



- After five steps.



- After six steps.



- There will be no more updates.

- Once the values have stabilised, we have a program for the reactve agent.
- At each step we pick the state with the highest utility.
- Again (as with situated automata) we can push the computation off-line.
  - Online the agent only needs a look-up table.
- We can also compute utilities online, as in reinforcement learning.

Hybrid Architectures

- Many researchers have argued that neither a completely deliberative nor completely reactive approach is suitable for building agents.
- They have suggested using *hybrid* systems, which attempt to marry classical and alternative approaches.
- An obvious approach is to build an agent out of two (or more) subsystems:
  - a *deliberative* one, containing a symbolic world model, which develops plans and makes decisions in the way proposed by symbolic AI; and
  - a *reactive* one, which is capable of reacting to events without complex reasoning.

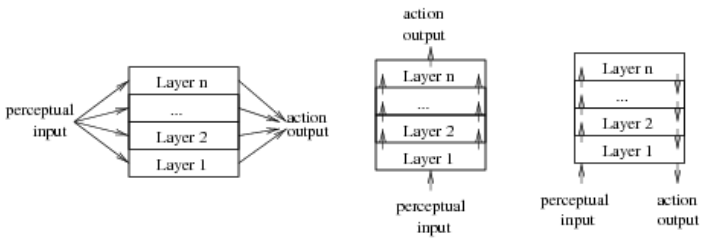
- Often, the reactive component is given some kind of precedence over the deliberative one.
- This kind of structuring leads naturally to the idea of a *layered* architecture, of which TOURINGMACHINES and INTERRAP are examples.
- In such an architecture, an agent's control subsystems are arranged into a hierarchy, with higher layers dealing with information at increasing levels of abstraction.

- A key problem in such architectures is what kind control framework to embed the agent's subsystems in, to manage the interactions between the various layers.
- *Horizontal layering*.

  Layers are each directly connected to the sensory input and action output.

  In effect, each layer itself acts like an agent, producing suggestions as to what action to perform.
- *Vertical layering*.

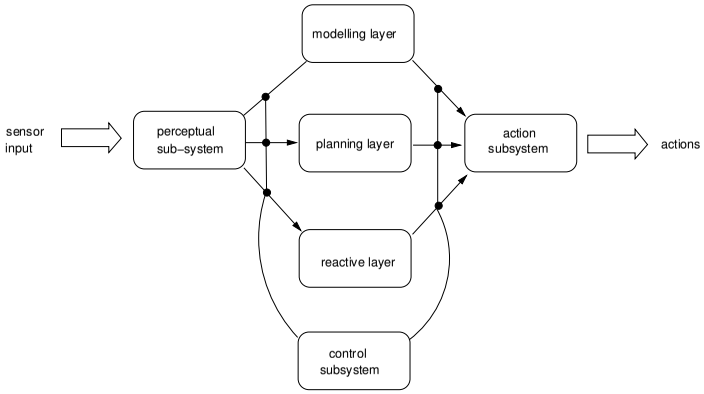
  Sensory input and action output are each dealt with by at most one layer each.



(a) Horizontal layering (b) Vertical layering (One pass control) (c) Vertical layering (Two pass control)

Ferguson — TOURINGMACHINES

- The TOURINGMACHINES architecture consists of *perception* and *action* subsystems, which interface directly with the agent's environment, and three *control layers*, embedded in a *control framework*, which mediates between the layers.
- A horizontally layered architecture.



- The *reactive layer* is implemented as a set of situation-action rules, *à la* subsumption architecture.

  Example:

```
rule-1: kerb-avoidance
    if
        is-in-front(Kerb, Observer) and
        speed(Observer) > 0 and
        separation(Kerb, Observer) < KerbThreshHold
    then
        change-orientation(KerbAvoidanceAngle)
```

- The *planning layer* constructs plans and selects actions to execute in order to achieve the agent's goals.

- The *modelling layer* contains symbolic representations of the 'cognitive state' of other entities in the agent's environment.
- The three layers communicate with each other and are embedded in a control framework, which use *control rules*.

  Example:

```
censor-rule-1:
    if
        entity(obstacle-6) in perception-buffer
    then
        remove-sensory-record(layer-R, entity(obstacle-6))
```

- Such control structures have become common in robotics.

Summary

- This lecture has looked at three kinds of agent:
  - Deductive reasoning agents;
  - Reactive agents; and
  - Hybrid agents.
- We looked at building deductive reaosning agents out of logic, and several logic-like languages.

  (Note that noting we looked at had very complex behavior)
- Reactive agents build complex behaviour from simple components.
- Hybrid agents try to combine the speed of reactive agents with the power of deliberative agents.
- Papers describing more complex logic specifications can be found on the webpage.