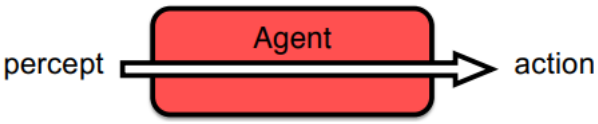
Contents

1 Limitations of simple reactive architecture

no representation of the environment means

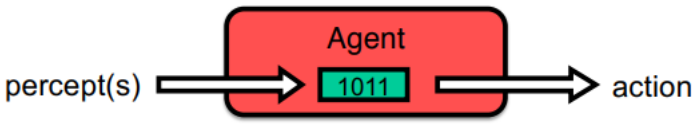
- its knowledge of the world is limited by the **range of its sensors**
- its unable to count (as opposed to recognise number)
- its unable to recover from actions which fail silently and many others

2 Modelling reactive behaviours



- we can model reactive behaviours as condition-action rules
- if the condition matches the agents precepts, it triggers an action `if percept then action`
- a simple reactive agent maintains no internal representation of the state of the world, whether the rule has been fired before etc.

3 Reactive architectures with state



- some rules match against an internal representation of aspects of the environment
- representations can be built using simple **percept-driven rules** (internal actions) which record simple beliefs about the state of the world

4 State

- we have simply taken a condition-action rule which matched against a percept and generated an action and split it in two, with a mediating internal representation
- needs extra machinery to store the state
- requires at least **two computation steps** to choose an action rather than one

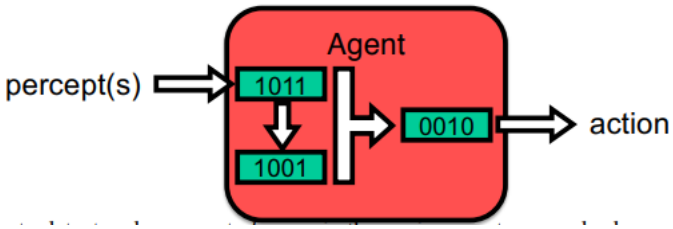
4.1 Actions which modify the internal state

- we can extend this to rules
  - whose conditions match against the agents internal state; and
  - whose actions modify the agents internal state
- again, this appears to make things worse: requires even **more space and more steps** to choose an action

4.2 Importance of representations

- notion of a rule which only responds to and generates internal changes in the agent is a key step
- forms the basis of all derived representations, and of representations which refer to other aspects of the agents internal state
- e.g., allows the agent to respond only to changes in the environment, ignoring features that are constant
- without some representation of the previous state, we cant say what is novel in the current state

5 Detecting change



to detect and represent changes in the environment we need rules

- whose conditions match against representations of the current and previous precepts
- whose action is to remember (copy) the state representing the current percept for use at the next cycle

5.1 Internal representations

- require more space and incur the cost of maintaining the representation
- allow the choice of actions based on sequences of states, e.g.: to react to change or **lack of it**
- given such internal behaviours, much more complex external behaviours are possible

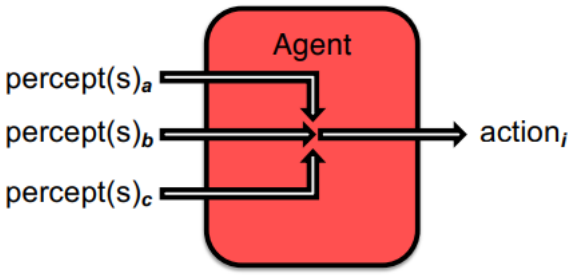
6 Action selection function

- the action selection function for a reactive agent with state looks like
  - $selectAction : (Event, State) \rightarrow (Action, State)$
- a reactive agent with state (finite-state machine) can respond to regular sequences of events
- we can add more complex state data structures to increase the capabilities of the agent, e.g.,
  - add a stack (pushdown automata) to respond to context-free sequences
  - add a random-access array to get a Turing machine, etc.

7 Advantages of state

- agents knowledge of the world is **no longer limited** by the range of its sensors, it can remember parts of the environment it cant currently sense
- agent is **able to count**, allowing it to execute behaviours that require some action to be **iterated** a given number of times
- agent is able to **recover** from actions which **fail silently**, it can remember which actions it has tried before or how many times an action has been tried, and **try something else** if the action doesnt have the desired effect

7.1 Combined actions



distinct actions triggered by different percepts are combined into a **single composite action**

7.2 Problems with combining actions

- action selection based on combining actions is **prone to a number of problems**:
  - **local minima**: e.g., Braitenberg vehicles get suck between to obstacles, unable to turn around
  - **cyclic behaviour**: e.g., Braitenberg vehicles get trapped orbiting an obstacle
- one way to solve these problems is simply to inject *noise* or *randomness* into the agents behaviours

7.3 Cyclic behaviour

Occurs when an agent repeats it's actions creating an infinite loop.

7.4 Local minima

consider a simple boids-like agent which chooses an action by combining

- a vector towards the goal
- a vector away from an obstacle (if any)

if there is an obstacle on the way to the goal, the agent can get stuck. Image a scenario where the goal is straight line from the agent and there is a single obsticle in front of it. One vector would try to move towards the goal, where the other would push against the obsticle. Both of them cancel each other out, leaving the agent stuck in a single place.

7.5 Escaping local minima with random vector

- adding a (small) random vector can take the agent past the obstacle, "unsticking" it. It would allow for the machine to move slightly while two initial vectors cancel each other out.
- However this would not work if the obsticle surounds the agent.

7.6 Avoid Past behaviour

- Avoid Past behaviour uses a short term representation of where the agent has been recently **stored in memory**
- repulsive forces are generated from recently visited areas
- the output of the Avoid Past behaviour is a vector of the same form as the vectors produced by the other behaviours (e.g., move to goal, avoid obstacles etc.) and is combined in the same way
- memory used by Avoid Past is **short term**. The agent **can return** to previously visited locations when the memory decays

This however still have a problem. The memory is short term, meaning that nothing prevents the agent from repeating the same mistake after a while. It could get stuck in *cycling behaviour*

7.7 Subsumption architecture

- **collection of behaviours** specified as a set of rules in Behavior Language
- behaviours compile to an Augmented Finite State Machine (AFSM)
- each AFSM performs an action and is responsible **for its own perception of the world**
- the output of one behaviour can **form the input to another**

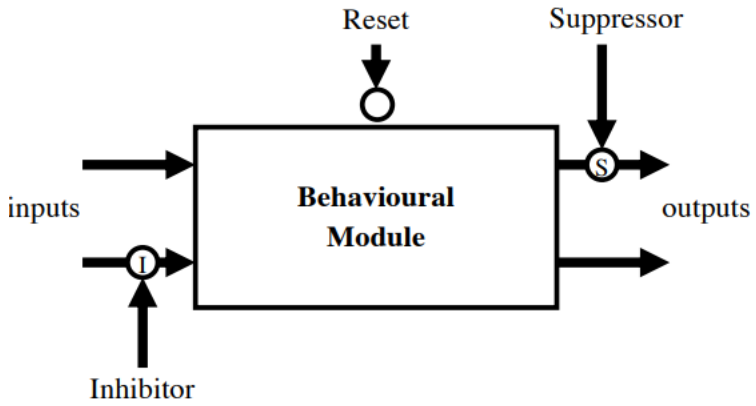


Figure 1: Behavioural model

- behaviours are organised into layers, each of which is responsible for **independently achieving a goal**
- complex actions subsume simpler behaviours lower in the hierarchy
- output of lower layers **can be read by higher layers**
- lower layers have **no knowledge** of higher layers
- layers operate **concurrently and asynchronously**
- higher layers control lower ones using:
  - **inhibition**: prevents transmission
  - **suppression**: replaces a message with a suppressing message
  - **reset**: restores behaviour to its original stat

7.8 Foraging

- behaviours are prioritised and the robot is executing only **one behaviour at any one time**
- when the robot senses an obstacle, **wandering is suppressed**, so that the avoidance behaviour can get the robot away from the obstacle
- when the robot senses the target object, **collision avoidance is suppressed**, otherwise the pickup behaviour couldnt get the robot close enough to the object to pick it up
- when the object is grabbed, homing then **suppresses pickup** (allowing the robot to ignore the potential distraction of of other objects it might encounter on its way back to base)

8 Advantages of stateful reactive architecture

- a reactive architecture with state can **produce any kind of behaviour** (no longer restricted to a fixed response to a given situation)
- no complex problem solving required
- if the state can be partitioned (e.g., subsumption architecture)
  - development is easier
  - can still use dedicated, parallel hardware
  - fast (real-time) response to changes in the environment

9 Disadvantages of stateful reactive architectures

- all responses must be defined in advance
- cant cope with novel situations for which they dont have a predefined behaviour
- agent programs for complex problems can be very large

Reference section

placeholder