REACTIVE AND HYBRID ARCHITECTURES

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 – behavior languages

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

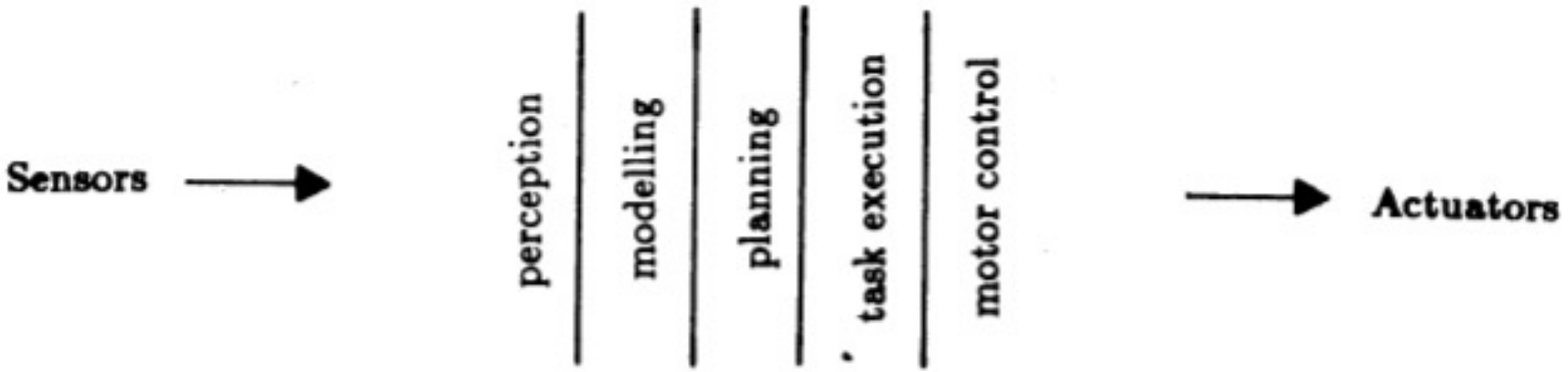
Brooks – behavior languages

- 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' behavior 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

Brooks – behavior languages

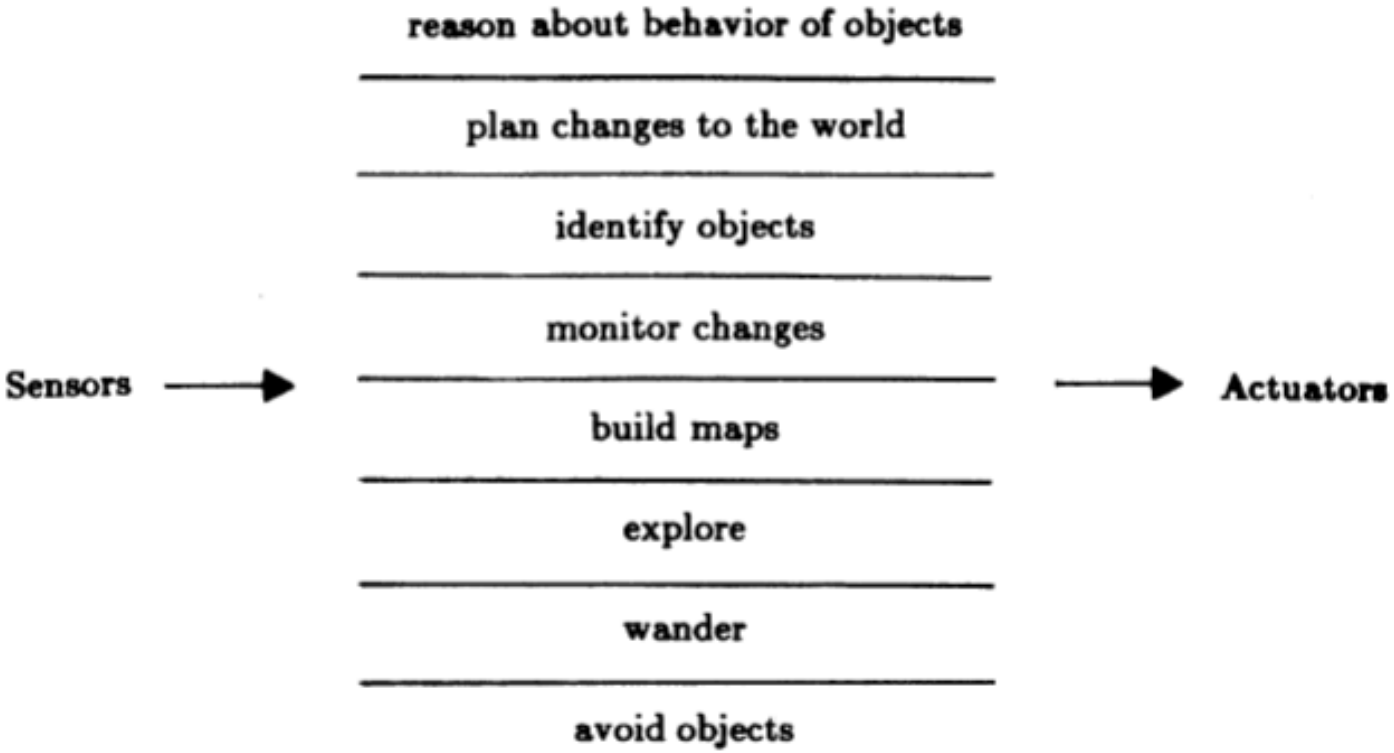
- To illustrate his ideas, Brooks built some based on his *subsumption architecture*
- A subsumption architecture is a hierarchy of task-accomplishing *behaviors*
- Each behavior is a rather simple rule-like structure
- Each behavior 'competes' with others to exercise control over the agent
- Lower layers represent more primitive kinds of behavior (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

A Traditional Decomposition of a Mobile Robot Control System into Functional Modules



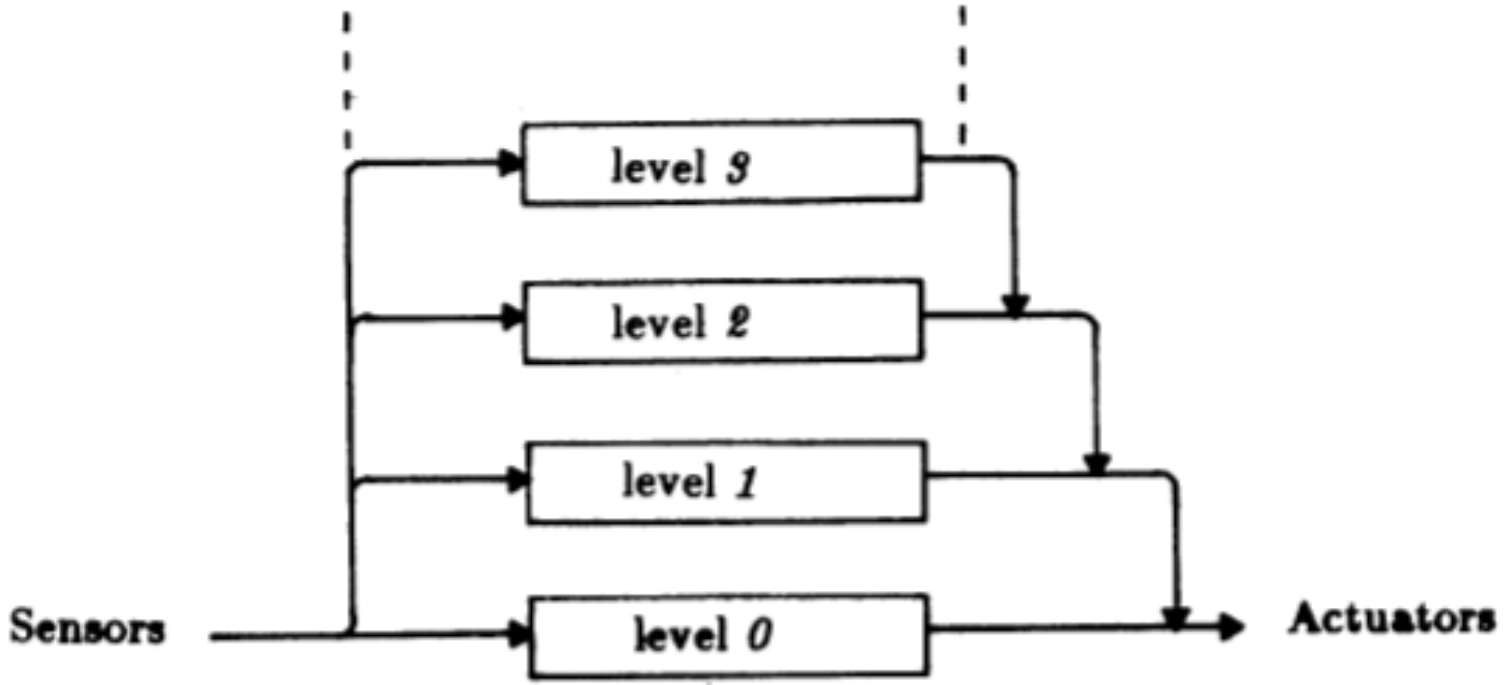
From Brooks, “A Robust Layered Control System for a Mobile Robot”, 1985

A Decomposition of a Mobile Robot Control System Based on Task Achieving Behaviors



From Brooks, "A Robust Layered Control System for a Mobile Robot", 1985

Layered Control in the Subsumption Architecture



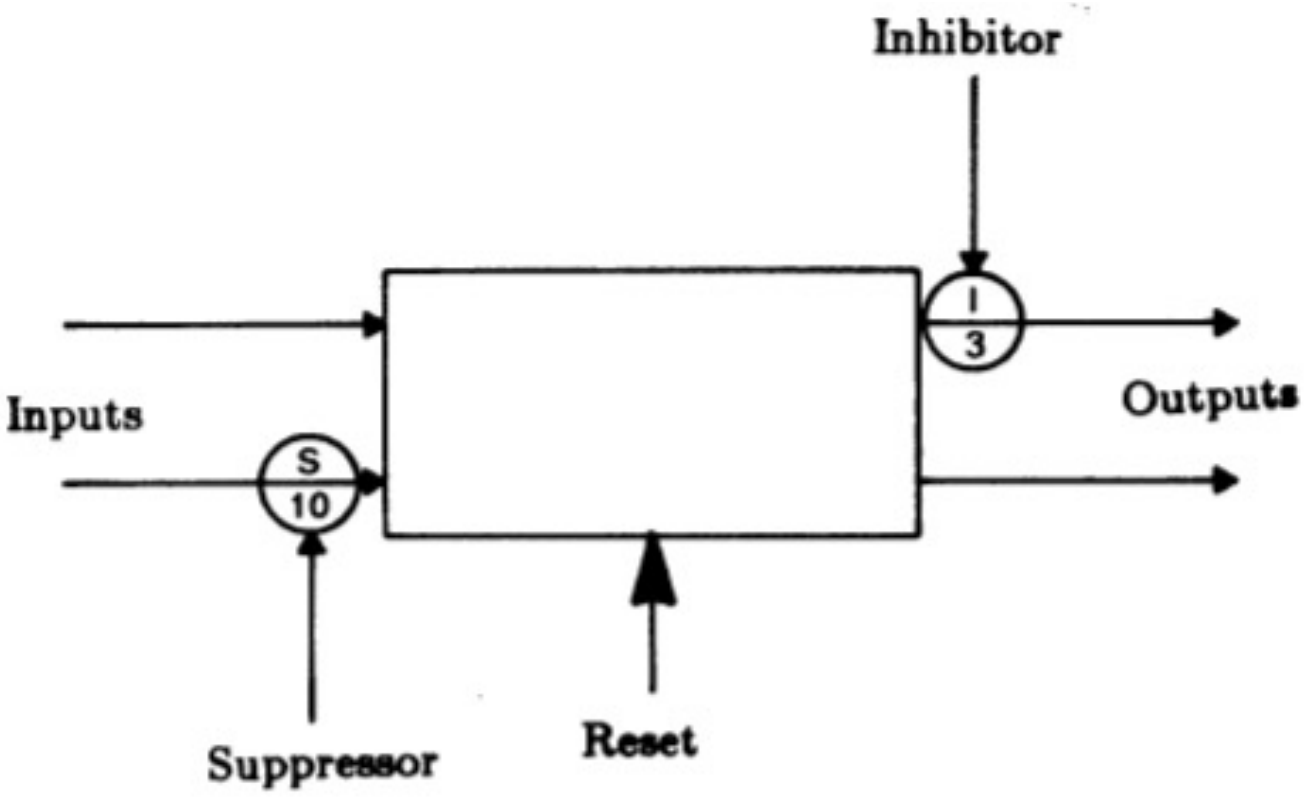
From Brooks, “A Robust Layered Control System for a Mobile Robot”, 1985

Example of a Module – Avoid

```
(defmodule avoid
  :inputs (force heading)
  :outputs (command)
  :instance-vars (resultforce)
  :states
    ((nil (event-dispatch (and force heading) plan))
     (plan (setf resultforce (select-direction force heading))
           go)
     (go (conditional-dispatch (significant-force-p resultforce 1.0)
                               start
                               nil))
     (start (output command (follow-force resultforce))
            nil)))
```

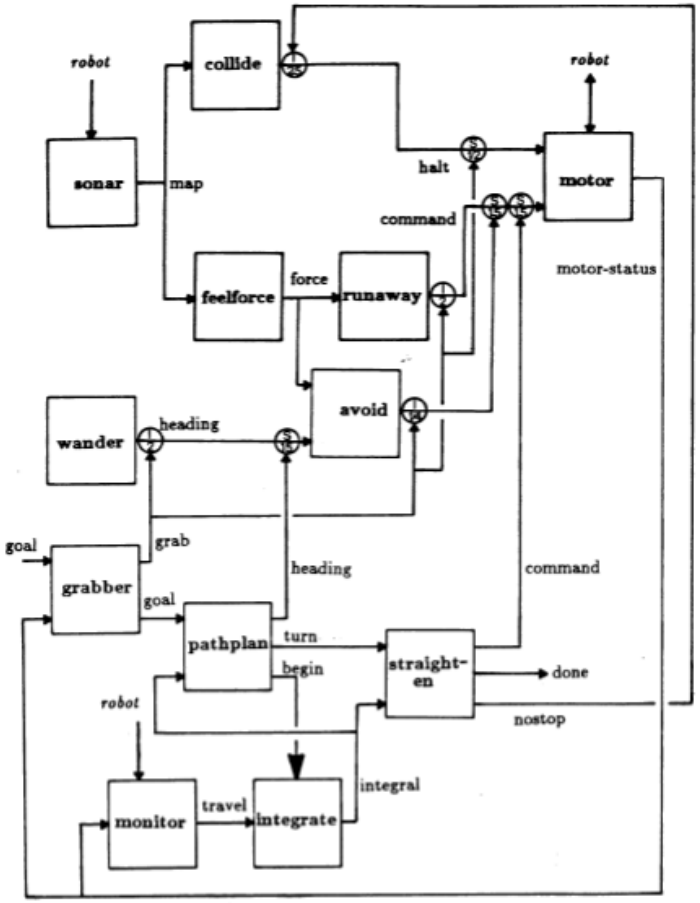
From Brooks, “A Robust Layered Control System for a Mobile Robot”, 1985

Schematic of a Module



From Brooks, “A Robust Layered Control System for a Mobile Robot”, 1985

Levels 0, 1, and 2 Control Systems



From Brooks, “A Robust Layered Control System for a Mobile Robot”, 1985

Steels' Mars Explorer

- 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.

Steels' Mars Explorer Rules

- 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 (1)

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

  if carrying samples *and* at the base
  then drop samples (2)

- Agents carrying samples will return to the mother-ship:

  if carrying samples and *not* at the base
  then travel up gradient (3)

Steels' Mars Explorer Rules

- Agents will collect samples they find:

  if detect a sample *then* pick sample up (4)

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

  if true *then* move randomly (5)

Situated Automata

- A sophisticated approach is that of Rosenschein and Kaelbling
- In their *situated automata* paradigm, an agent is specified in a rule-like (declarative) language, and this specification is 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*

Situated Automata

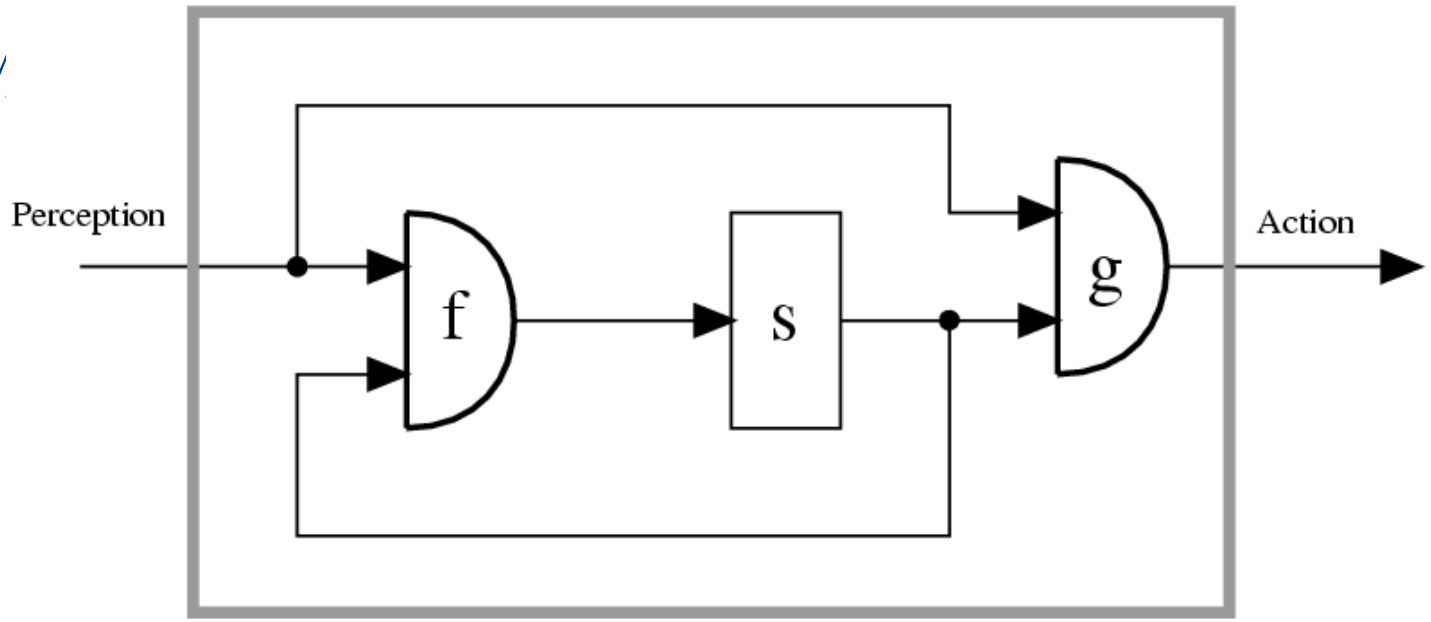
- The logic used to specify an agent is essentially a modal logic of knowledge
- The technique depends upon the possibility of giving the worlds in possible worlds semantics a concrete interpretation in terms of the states of an automaton
- "[An agent]…*x* is said to carry the information that *P* in world state *s*, written $s \models K(x,P)$, if for all world states in which *x* has the same value as it does in *s*, the proposition *P* is true."

  [Kaelbling and Rosenschein, 1990]

Situated Automata

- An agent is specified in terms of two components: perception and action
- Two programs are then used to synthesize agents
  - RULER is used to specify the perception component of an agent
  - GAPPS is used to specify the action component

Circuit Model of a Finite-State M



f = state update function
s = internal state
g = output function

From Rosenschein and Kaelbling,
"A Situated View of Representation and Control", 1994

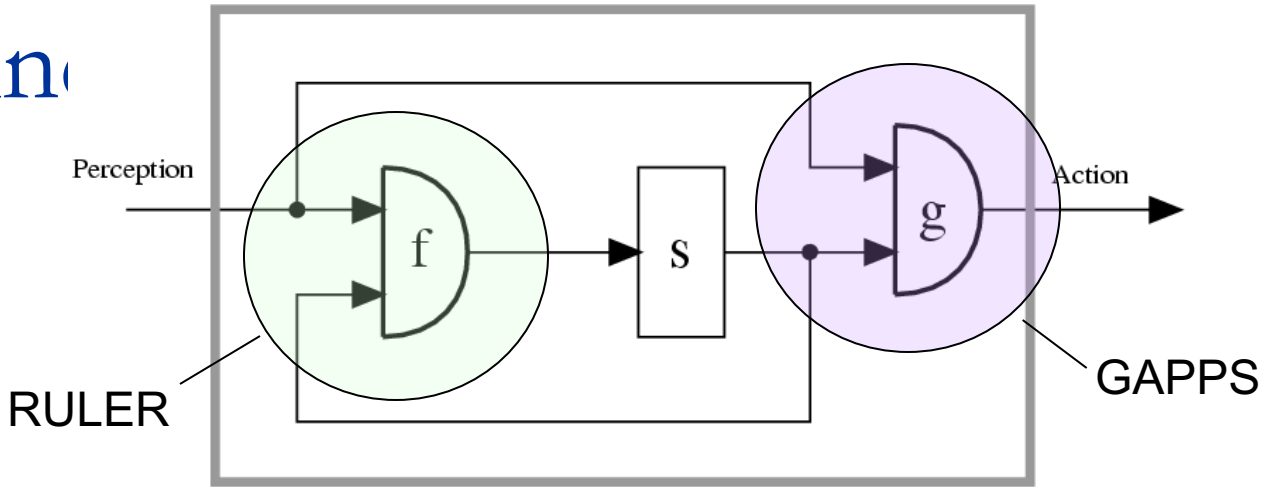
RULER – Situated Automata

- RULER takes as its input three components
- "[A] specification of the semantics of the [agent's] inputs ('whenever bit 1 is on, it is raining'); a set of static facts ('whenever it is raining, the ground is wet'); and a specification of the state transitions of the world ('if the ground is wet, it stays wet until the sun comes out'). The programmer then specifies the desired semantics for the output ('if this bit is on, the ground is wet'), and the compiler ... [synthesizes] a circuit whose output will have the correct semantics. ... All that declarative 'knowledge' has been reduced to a very simple circuit." [Kaelbling, 1991]

GAPPS – Situated Automata

- The GAPPS program takes as its input
  - A set of *goal reduction rules*, (essentially rules that encode information about how goals can be achieved), and
  - a top level goal
- Then it generates a program that can be translated into a digital circuit in order to realize the goal
- The generated circuit does not represent or manipulate symbolic expressions; all symbolic manipulation is done at compile time

Circuit Model of a Finite-State Machin



"The key lies in understanding how a process can naturally mirror in its states subtle conditions in its environment and how these mirroring states ripple out to overt actions that eventually achieve goals."

From Rosenschein and Kaelbling,
"A Situated View of Representation and Control", 1994

Situated Automata

- 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.)

Advantages of Reactive Agents

- Simplicity
- Economy
- Computational tractability
- Robustness against failure
- Elegance

Limitations of Reactive Agents

- Agents without environment models must have sufficient information available from local environment
- If decisions are based on *local* environment, how does it take into account *non-local* information (i.e., it has a "short-term" view)
- Difficult to make reactive agents that learn
- Since behavior emerges from component interactions plus environment, it is hard to see how to *engineer* specific agents (no principled methodology exists)
- It is hard to engineer agents with large numbers of behaviors (dynamics of interactions become too complex to understand)

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
  - a *reactive* one, which is capable of reacting to events without complex reasoning

Hybrid Architectures

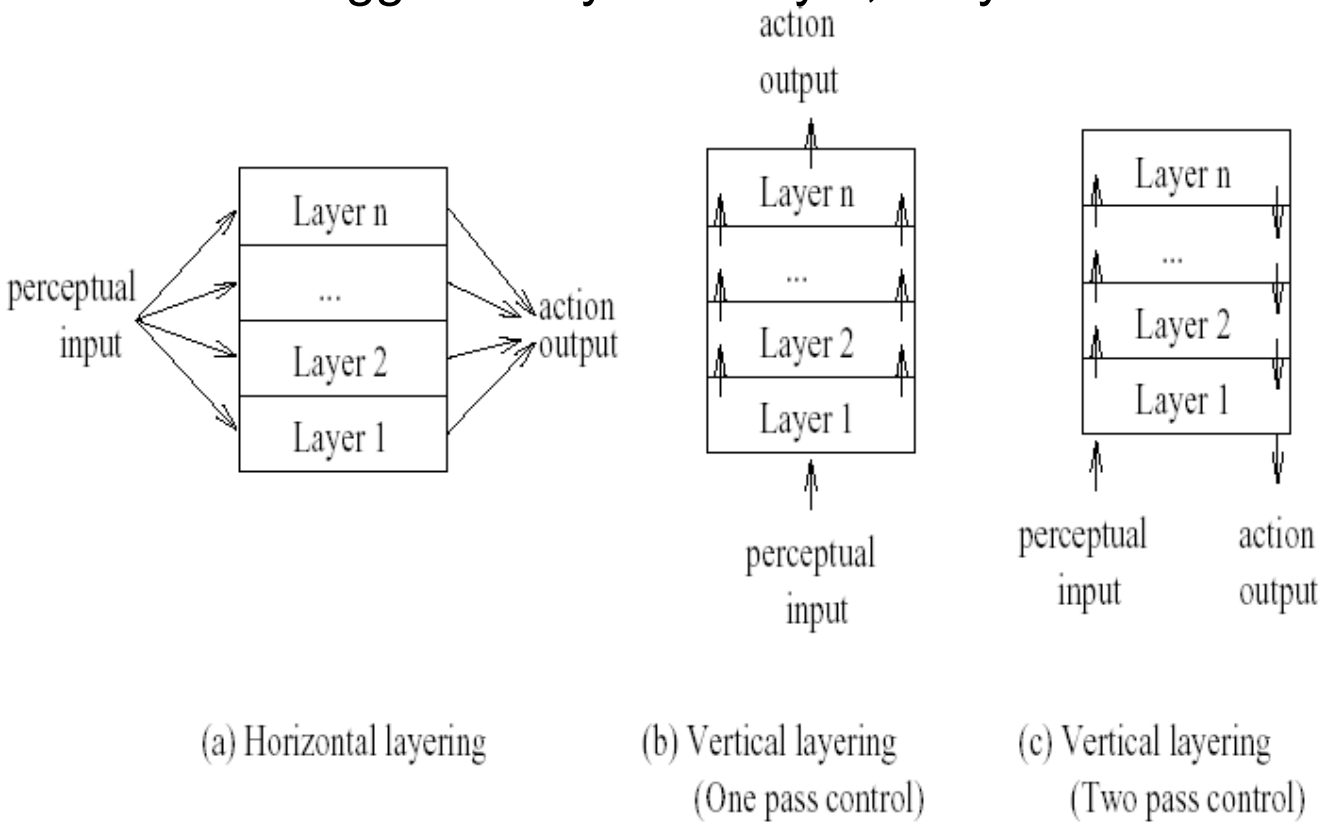
- Often, the reactive component is given some kind of precedence over the deliberative one
- This kind of structuring leads naturally to the idea of a *layered* architecture, of which TOURINGMACHINES and INTERRAP are examples
- In such an architecture, an agent's control subsystems are arranged into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

Hybrid Architectures

- A key problem in such architectures is what kind of 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

Hybrid Architectures

m possible actions suggested by each layer, *n* layers



(a) Horizontal layering

(b) Vertical layering (One pass control)

(c) Vertical layering (Two pass control)

m^n interactions

Introduces bottleneck in central control system

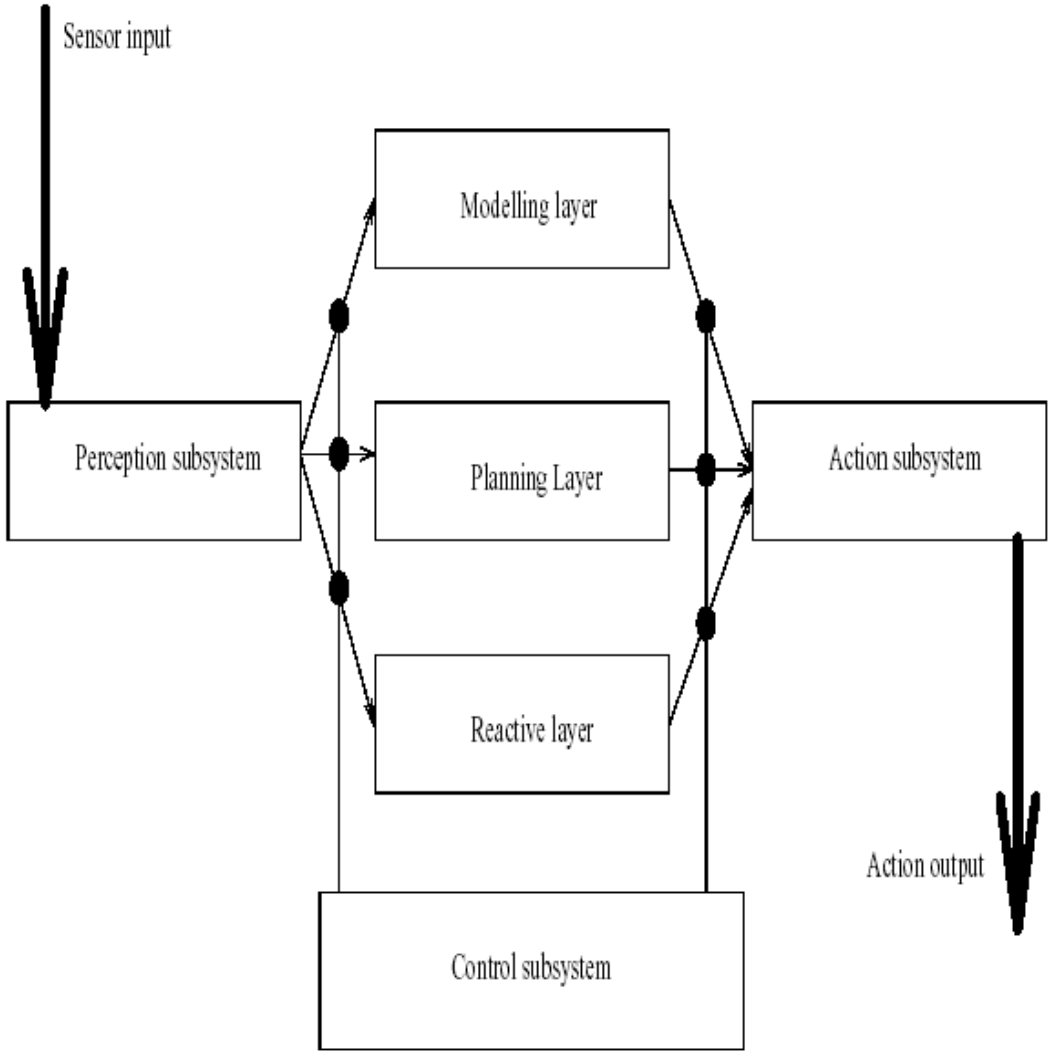
$m^2(n-1)$ interactions

Not fault tolerant to layer failure

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

Ferguson – TOURINGMACHINES



Ferguson – TOURINGMACHINES

- The *reactive layer* is implemented as a set of situation-action rules, *a 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

Ferguson – TOURINGMACHINES

- The *modeling 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))
```

Müller –InteRRaP

- Vertically layered, two-pass architecture