Introduction to Multi-Agent Programming

3. Fundamental Agent Architectures

Logic-Based, Reactive, and Hybrid Architectures, CS-Freiburg Case Study

Alexander Kleiner, Bernhard Nebel

Contents

Introduction
History of development

1956-1985: Originally agents were mainly based on *symbolic reasoning*

- Researches concluded the weakness of this approach for time-constrained domains

1985-present: Research on *reactive agents*

- Decision making based on syntactic manipulation of the representation
- The idea that intelligent behavior is seen as innately linked to the environment an agent occupies - intelligent behavior is not disembodied, but is a product of the interaction the agent maintains with its environment
- the idea that intelligent behavior emerges from the interaction of various simpler behaviors

From 1990-present: a number of alternatives proposed: *hybrid* architectures, which attempt to combine the best of reasoning and reactive architectures

Logic-Based Architectures
Introduction

- Traditional approach of building AI systems, known as symbolic AI
  - Contains an explicitly represented, symbolic model of the world
  - The state of the world is represented by a database of predicates
    - Open(valve221)
    - Temperature(reactor4726,321)
  - Makes decisions about what actions to perform via symbolic reasoning, e.g., logical deduction or theorem proving
  - Idea that intelligent behavior can be generated by such representation and manipulation of symbols

Logic-Based Architectures

Formal Model

- Action selection by 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
  - ***A*** be the set of actions the agent can perform
  - $\Delta, \rho \vdash \phi$ mean that ϕ can be proven from Δ using ρ
- We assume the automatic execution of the functions
  - *see*(s), which returns percepts according to the current world state (not one-to-one!)
  - *next*(Δ, p), which updates the data base according to new percepts

Logic-Based Architectures

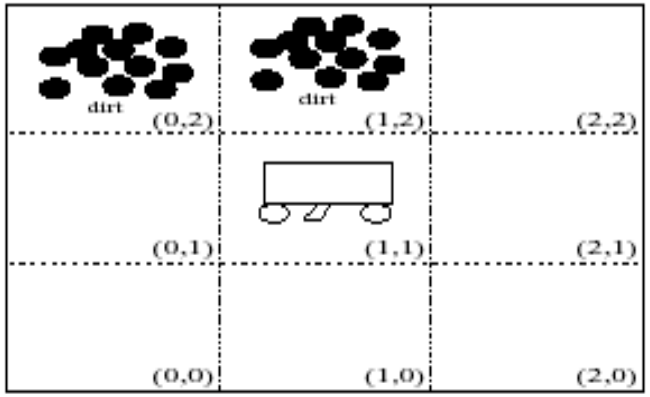
Action Selection Algorithm

```
function action (Δ∈D): A {
   //try to find an action explicitly prescribed
   for each a∈A do {
      if Δ, ρ ⊢ Do(a) then
      then return a
   }

   // try to find an action not excluded
   for each a∈ A do {
      if Δ, ρ ⊬ ¬Do(a) then
      then return a
    }
  return NULL
}
```

Logic-Based Architectures

Example: Vacuum World (1)

- Cleaning robot with
  - percepts P = *{dirt, X,Y,θ}*
  - Actions *A* = *{turnRight, forward, suck}*
- *Start: (0,0,North)*
- Goal: searching and cleaning dirt



Use of domain predicates to solve problem:

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*

Logic-Based Architectures

Example: Vacuum World (2)

- Set of rules *p* for solving the problem:
  - In(x,y) ∧ Dirt(x,y) → Do(suck)
  - In(0,0) ∧ Facing(north) ∧ ¬Dirt(0,0) → Do(forward)
  - In(0,1) ∧ Facing(north) ∧ ¬Dirt(0,1) → Do(forward)
  - In(0,2) ∧ Facing(north) ∧ ¬Dirt(0,2) → Do(turn)
  - In(0,2) ∧ Facing(east) → Do(forward)
  - …
- In order to ensure always one single action, ¬Dirt(X,Y) has to be explicitly checked

Logic-Based Architectures

Example: Vacuum World (3)

- Advantages
  - Pro-active behavior (deliberation)
  - Elegant logical semantics
- Problems:
  - How to convert video camera input to *Dirt*(0, 1)?
  - Time complexity for reasoning
    - Time for reasoning can be non-instantaneous
    - During computation, the dynamic worlds might change and thus the solution not valid anymore!
    - How to represent temporal information, e.g., how a situation changes over time?

Plan-based Architectures

- A variation of logic-based architectures
- A planning system is responsible for generating action sequences in order to reach the goals
- Special-purpose reasoning system geared towards generating plans and based on that selecting the right action
- First system: STRIPS (in the 70’s)
- These days: Numerous systems (e.g. our system Fast Downward and extensions)

BDI Architectures
Introduction

- Motivated from philosophy: theory of practical reasoning requires belief, desire, and intention (BDI)

- Successful example: Procedurale Reasoning System (PRS) (Georgeff & Lansky 1987)

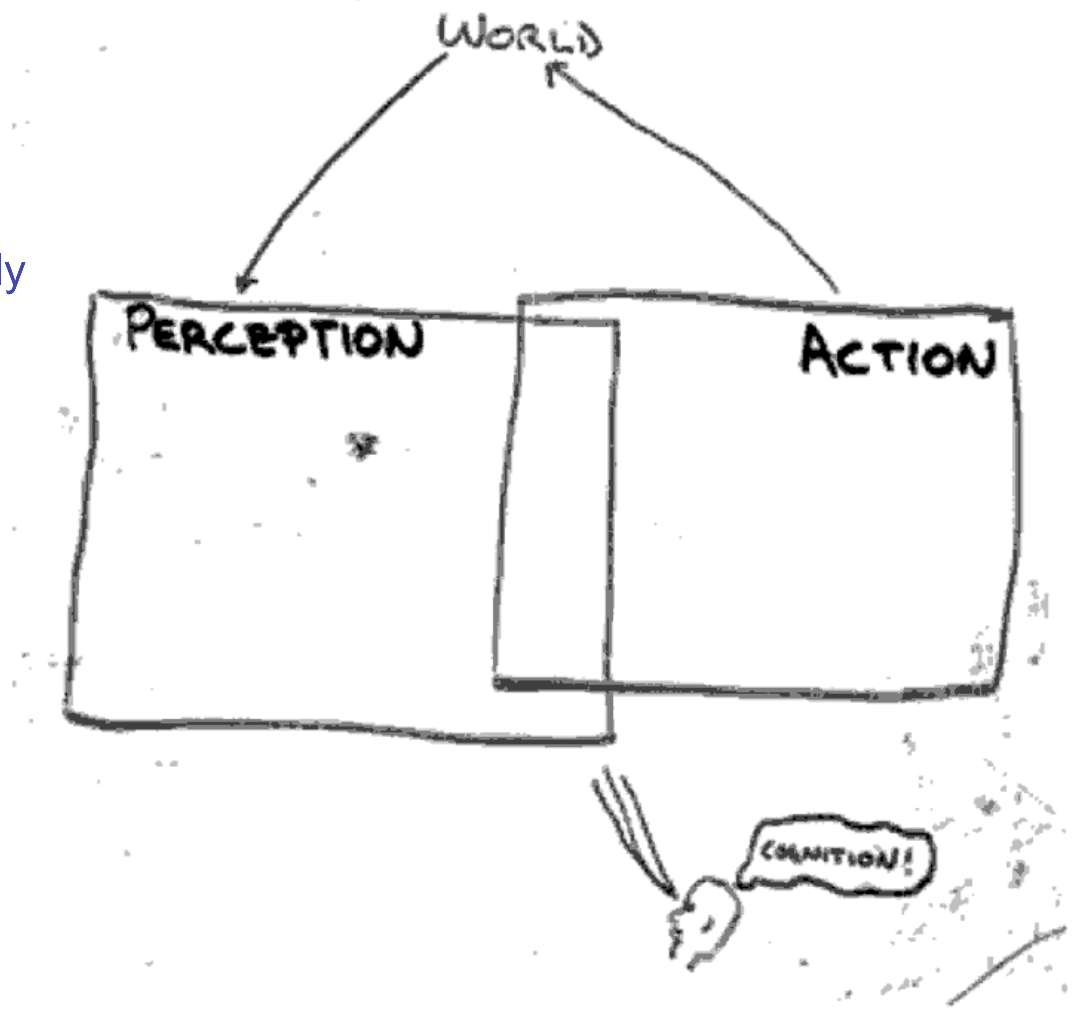
Reactive Architectures

Brooks: Subsumption Architecture

- Brooks´ Vision:
  - Intelligent behaviour can be generated *without* explicit representations of the kind that symbolic AI proposes
  - Intelligent behaviour can be generated *without* explicit abstract reasoning of the kind that symbolic AI proposes
  - Intelligence is an *emergent* property of certain complex systems
- Two key ideas:
  - Situatedness and embodiment. 'Real' intelligence is situated in the world, not in disembodied systems such as theorem provers or expert systems.
  - 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.

Subsumption Architecture
Brooks' Vision (1)

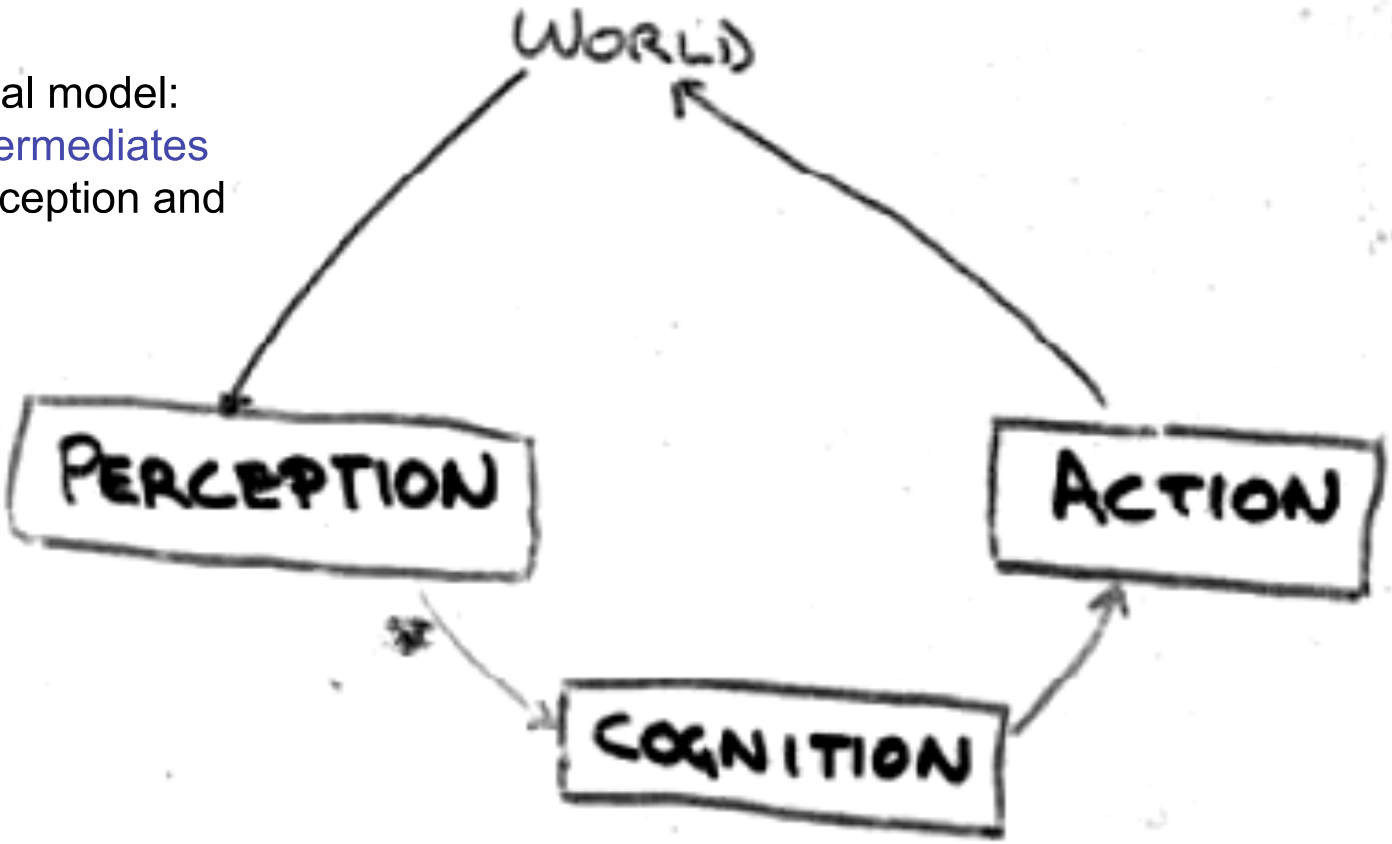
The new model: perception and action is all there is. Cognition is only in the eye of the observer.



Original slides from R. Brooks held at the seminar “From Pixels to Predicates” (1983)

Subsumption Architecture
Brooks' Vision (2)

The traditional model: cognition intermediates between perception and action



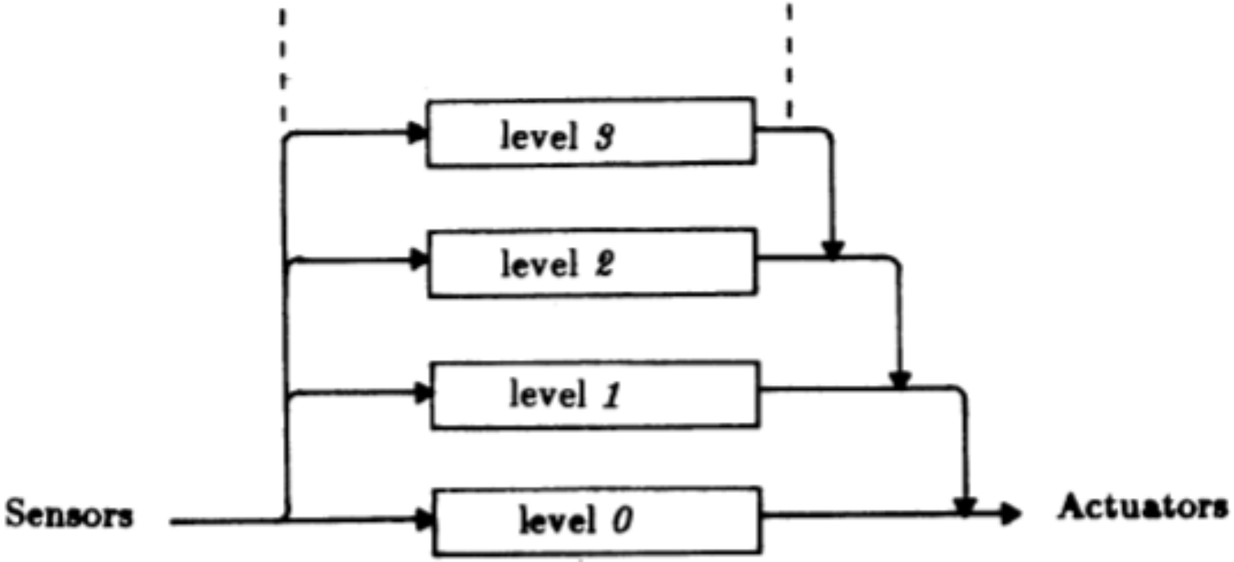
Original slides from R. Brooks held at the seminar "From Pixels to Predicates" (1983)

Subsumption Architecture
Behaviors and Layered control

- Decision making by a set of task accomplishing behaviors
  - Behaviors are direct mappings from observations to actions
    - Processing of raw sensor data
    - Direct coupling between observation and action, e.g. light switch pressed → light on
    - Behaviors implemented as asynchronous finite state machines
- Multiple behaviors can “fire” simultaneously
  - mechanism for action selection: subsumption hierarchy
  - Behaviors organized in layers
    - Higher layer behaviors inhibit lower level ones
    - E.g., „Avoid obstacles“ lower layer (higher priority) than „drive to goal“
- Note behaviors are acting independently and communicate asynchronously, however are not pro-active as in MAS

Subsumption Architecture

Layered Control



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

For Example:

- Level0: Avoid Obstacles
- Level1: Wander aimlessly around
- Level2: Heading towards goals points
- Level3: Select unexplored locations as goals

Subsumption Architecture

Formal Model

- A behavior $b \in Beh$ is (c, a) with $c \subseteq P, a \in A$, where *P* is the set of percepts and *A* the set of actions

- A behavior fires if the environment is in state $s \in S$ and iff $see(s) \in c$

- The subsumption hierarchy is implemented by the inhibition relation $b_1 \prec b_2$, denoting "b_1 inhibits b_2"

Subsumption Architecture

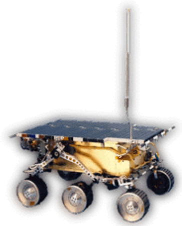
Action Selection Algorithm

```
function action (s ∈ S) : A  {

    // Compute the set of firing behaviors
    FB = {(c,a) | (c,a) ∈ Beh ∧ see(s) ∈ c}

    // find action with highest priority
    for each (c,a) ∈ FB do
    {
     if ¬(∃(c',a') ∈ FB such that (c',a') ≺ (c,a))
     then return a
    }
    return NULL

}
```

→ Time complexity: $O(n^2)$

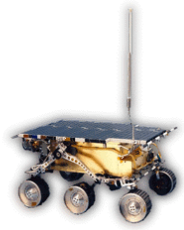
Subsumption Architecture

Steels' Mars Explorer Experiment (1)



- Steels 1990: Task of exploring a distant planet, more concretely, to collect samples of a particular type of precious rock.
  - The location of the rock samples is not known in advance, but they are typically clustered in certain spots.
  - A number of autonomous vehicles are available that can drive around the planet collecting samples and later reenter a mother ship spacecraft to go back to Earth.
  - There is no detailed map of the planet available
  - No communication between the vehicles due to obstacles, such as hills, valleys, etc.
- Solution idea
  - Gradient field: Direction and distance to the mother ship can be computed from an emitted radio signal
  - Indirect communication: Robots release "radioactive crumbs" that can be detected by others (enables emergent behavior)

Subsumption Architecture
Steels' Mars Explorer Experiment (2)



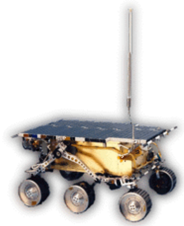
Individual agent's (goal-directed) behavior:

```
obstacle → changeDirection                                    (1)
carryingSamples ∧ atTheBase → dropSamples                      (2)
carrying Samples ∧ ¬ atTheBase → travelUpGradient              (3)
detectSample → pickUpSample                                    (4)
TRUE → moveRandomly                                            (5)
```

Subsumption hierarchy: (1) ≺ (2) ≺ (3) ≺ (4) ≺ (5)

Modification: Collaborative behavior: If sample is found, drop „crumb trail" while returning to ship (as guide for other agents (special rocks appear in clusters!). Other agents will weaken trail on way to samples. If sample cluster is empty → no trail reinforcement → trail „dies".

Subsumption Architecture
Steels' Mars Explorer Experiment (3)



Modification: Collaborative behavior:

```
obstacle → changeDirection                                      (1)
carryingSamples ∧ atTheBase → dropSamples                       (2)
carrying Samples ∧ ¬ atTheBase
   → drop_2_Crumbs ∧ travelUpGradient                           (3‘)
detectSample → pickUpSample                                     (4)
senseCrumbs → PickUp_1_Crumb ∧ travelDownGradient               (6)
TRUE → moveRandomly                                             (5)
```

subsumption hierarchy: (1) ≺ (2) ≺ (3‘) ≺ (4) ≺ (6) ≺ (5)

Subsumption Architecture
Pros and Cons (1)

- Pro
  - Simplicity, i.e. modules have high expressiveness
  - Computational tractability
  - Robustness against failure, i.e. possibility of modeling redundancies
  - Overall behavior emerges from interactions
- Cons
  - Behaviors are hard-coded with respect to the environment
  - Behavior emerges from interactions → How to engineer the system in the general case?
  - How to model long-term decisions?
  - How to implemented varying goals?
  - Design approach does no scale-up for large systems

Subsumption Architecture
Pros and Cons (2)

- In practice, the subsumption architecture is not sufficiently modular:

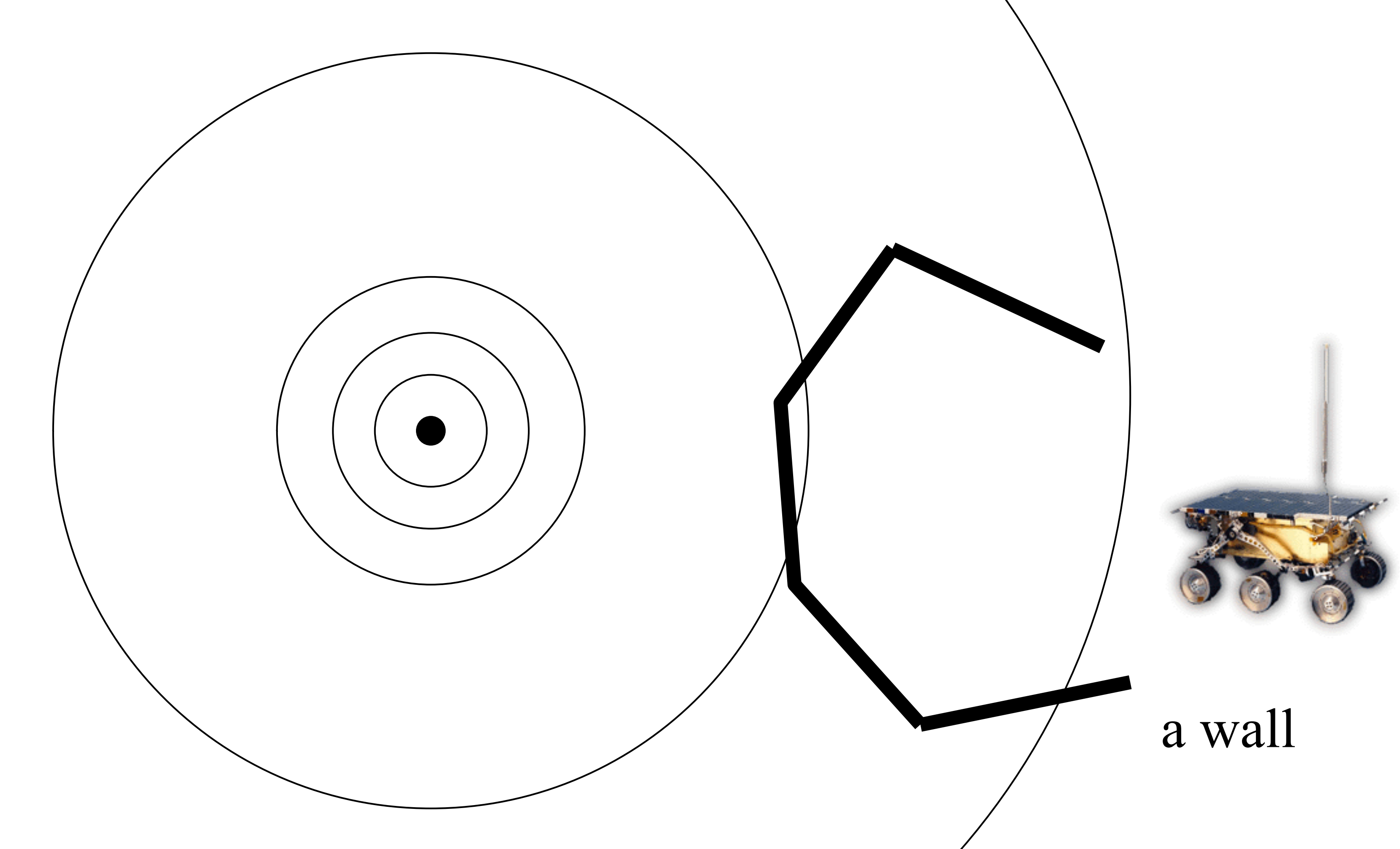
  … Because the upper layers interfere with the internal functions of lower-level behaviors, they cannot be designed independently and become increasingly complex. This also means that even small changes to low-level behaviors or to the vehicle itself cannot be made without redesigning the whole system….

  Hartley „Experiments with the Subsumption Architecture“, ICRA 1991

Subsumption Architecture

Pros and Cons (3)

- Is it here possible using the subsumption architecture for reaching the mother ship?



Behavior Networks
Introduction

- Composed of a set of *competence modules (Maes 1989)*
- Each module resembles behaviors like in the subsumption architecture
- Modules are defined
  - in terms of pre- and post-conditions (similar to STRIPS formalisms)
  - A real-value activation level (giving the relevance within particular situations)
- Modules are compiled into a spreading network accordingly

Behavior Networks
Definition

- ***P*** is a set propositional atoms generated from the world state

- Behavior networks are tuples (P, G, M, Π), where
  - **G** ⊆ P is the goal specification
  - **M** is a finite set of competence modules,

    where $m \in M$ is a tuple (*pre*, eff^{+}, eff^{-}, *beh*) with
  - ***pre*** ⊆ P denoting the preconditions
  - $\boldsymbol{eff^{+}}, \boldsymbol{eff^{-}}$ ⊆ P denoting the positive and negative effects (with $eff^{+} \cap eff^{-} = \emptyset$)
  - ***beh*** an executable behavior

Behavior Networks
Definition

- Competence modules are connected in a network; “activation energy” goes from goals to modules
- A *positive effect link* connects a positive effect p of a competence module to the precondition p of another competence module
- A *negative effect link* connects a negative effect p of one competence module to the precondition p of another competence module.

Behavior Networks
Activation flow

Module activation from situation

Activation of module *k* by satisfied preconditions $pre_k \cap S^t$, where M_p is the set of modules activated by *p* and $|pre_k|$ the number of *k*'s inputs.

$$\alpha^t_{k,e} = \phi \sum_{p \in pre_k \cap S^t} \frac{1}{|M_p| \cdot |pre_k|}$$

Fan effect

Input normalization

Module activation from goals

Activation by goals G_t satisfying positive effects eff^+ (or suppression from negative effects eff^- deleting goal propositions R^t that are already active), where N_e is the set of modules generating effect *e*.

$$\alpha^t_{k,gp} = \gamma \sum_{e \in eff_k^+ \cap G^t} \frac{1}{|N_e| \cdot |eff_k^+|}$$

$$\alpha^t_{k,gn} = -\delta \sum_{e \in eff_k^- \cap R^t} \frac{1}{|N_e| \cdot |eff_k^-|}$$

Behavior Networks
Activation flow

Module activation from predecessors

Activation of module k from activated modules E, where *p* is input of *k* and also positive effect of predecessor *l*

$$\alpha_{k,p}^{t} = \frac{\phi}{\gamma} \sum_{l \in E \setminus \{k\}} \sum_{p \in \left(pre_k \cap eff_l^{+}\right) \setminus S^t} \frac{1}{|M_p| \cdot |pre_k|}$$

Module activation from successors

Activation of module *k* from effect *e* that satisfy precondition of successor *l*

$$\alpha_{k,s}^{t} = \sum_{l \in K \setminus E \setminus \{k\}} \sum_{e \in \left(eff_k^{+} \cap pre_l\right) \setminus S^t} \frac{\alpha_k^{t-1}}{|N_e| \cdot |pre_k|}$$

Overall activation of module k:

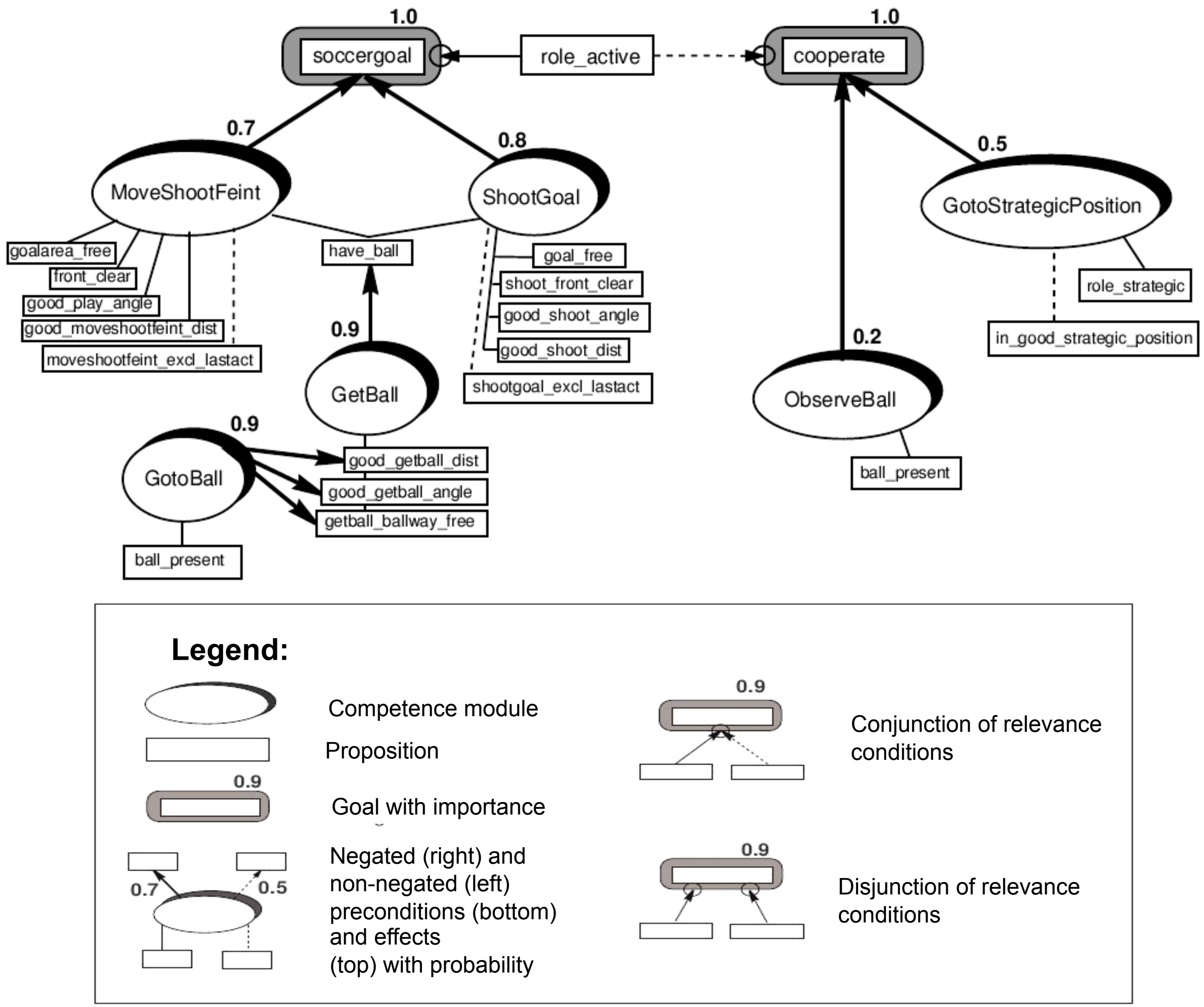
$$\alpha_{k,\Sigma}^{t} = \alpha_{k,e}^{t} + \alpha_{k,gp}^{t} + \alpha_{k,gn}^{t} + \alpha_{k,p}^{t} + \alpha_{k,s}^{t}$$

Behavior Networks

Action selection

1. Calculation of activation from goals end situation
2. Computation of inter-module activation
3. Uniform reduction of activation of each module to keep $\sum a_k$ constant
4. Select module with highest activation a_{best}
5. If $a_{best} > \theta$ then execute behavior
6. If not, reduce θ by 10%, restart at 1.)

Behavior Networks
Network example



Extended Behavior Networks (EBNs)

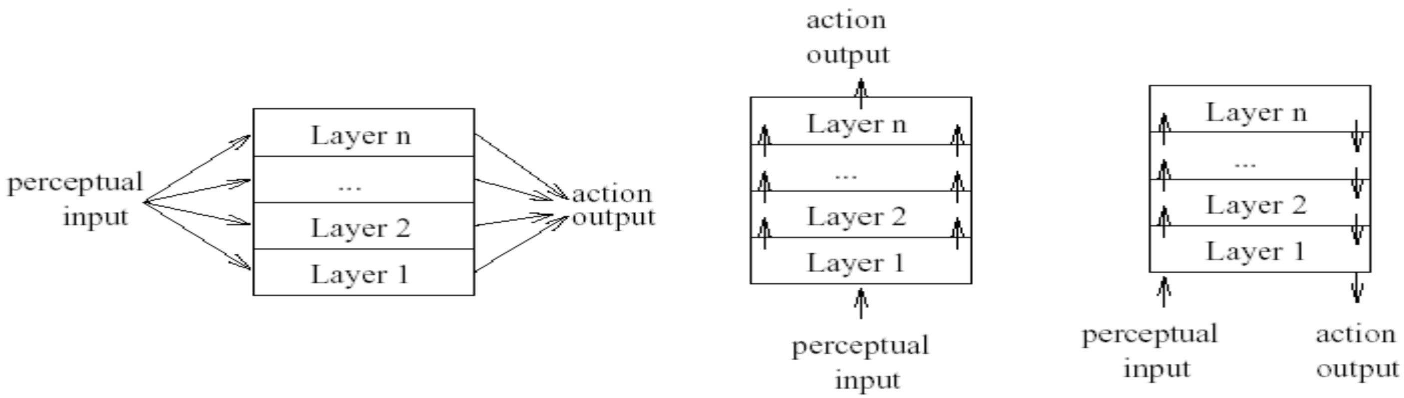
- Decision theoretic action selection
- Combine purely reactive acting with deliberation
- Modeling of continuous state variables
  - For example: “near goal”, goalDist= 1.2m
- No feedback loops
- No fan effect
- Computational more expensive

Hybrid Architectures
Introduction

- Neither completely deliberative nor completely reactive approaches are suitable for building agents
  - Researchers concluded using *hybrid* systems, which attempt to combine classical and alternative approaches
- An obvious approach is to build agents 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
- The combination of reactive and proactive behavior leads to a class of architectures in which the various subsystems are arranged into a hierarchy of interacting *layers*

Hybrid Architectures
Types of 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 (mostly used nowadays)



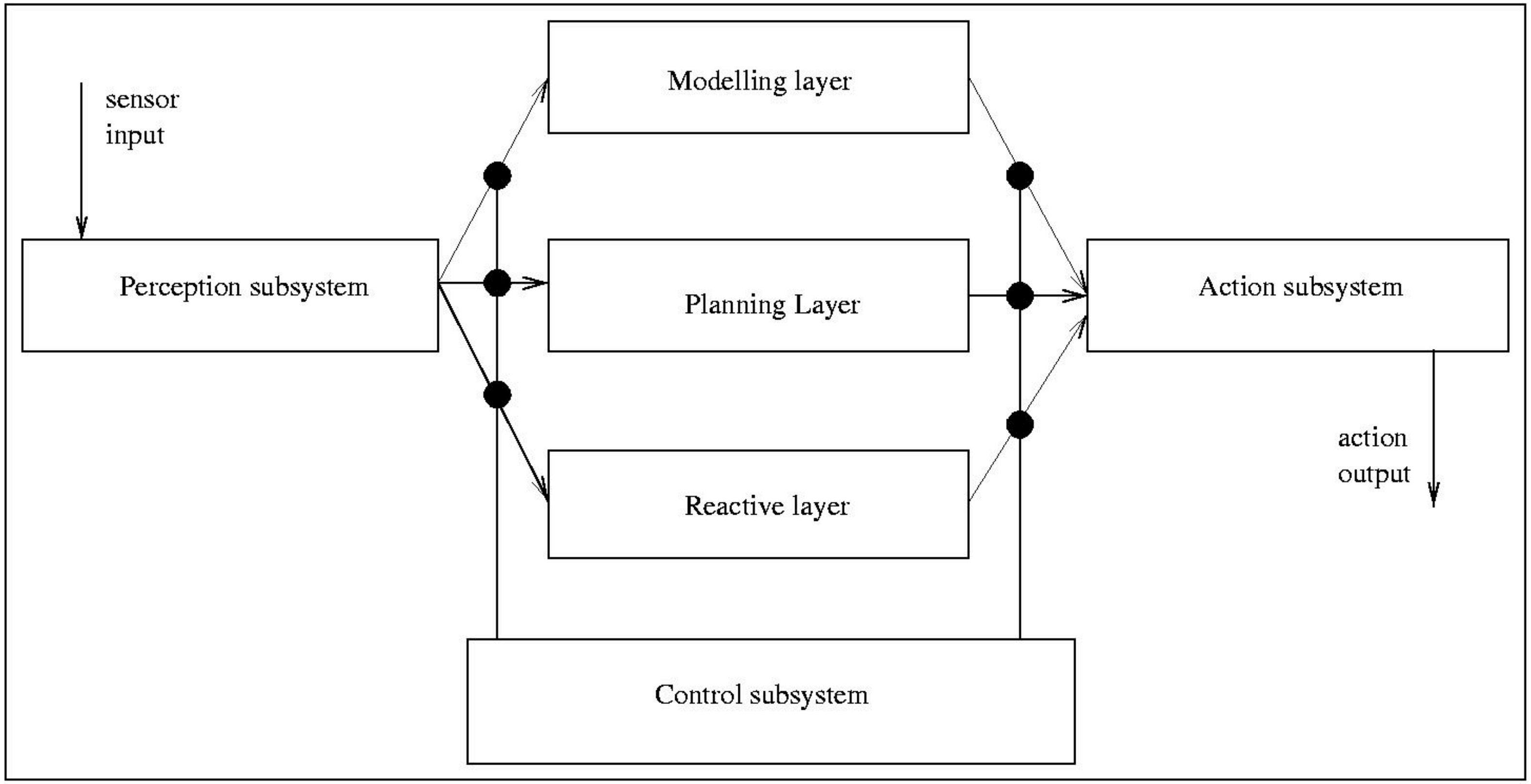
(a) Horizontal layering

(b) Vertical layering (One pass control)

(c) Vertical layering (Two pass control)

Hybrid Architectures

Example Horizontal Layering: “TouringMachines” (1)



(Ferguson 1992)

Hybrid Architectures

Example Horizontal Layering: “TouringMachines” (2)

- **Reactive Layer.** Subsumption-Architecture rules, e.g.:

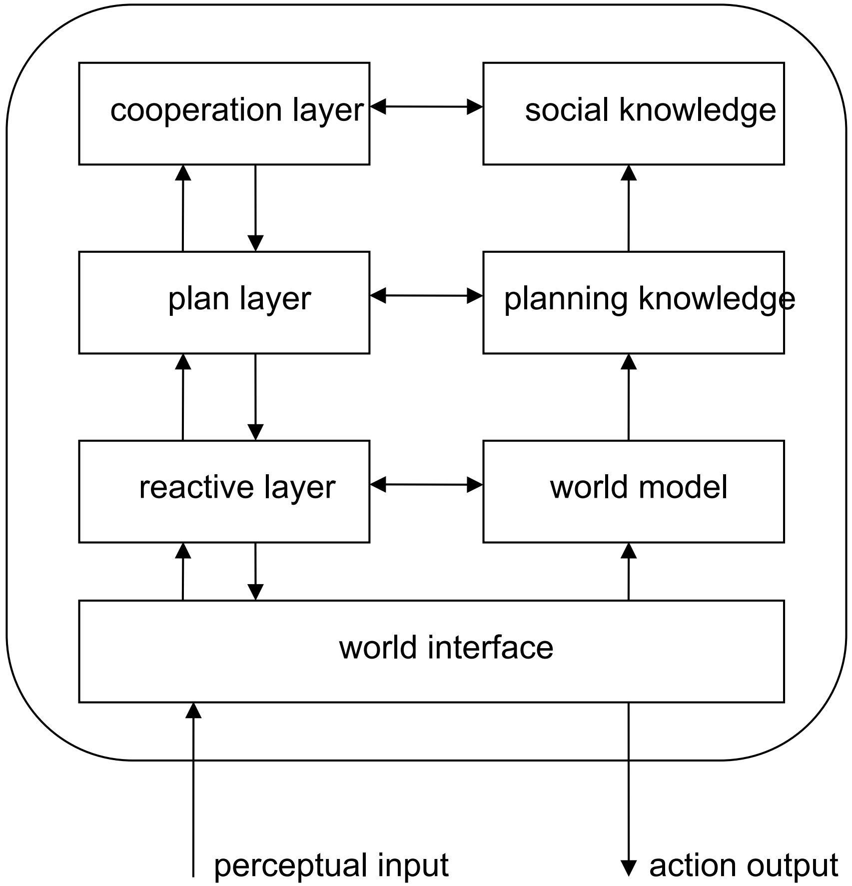
```
rule-1: kerb-avoidance
   if
       is-in-front(Kerb, Observer) and
       speed(Observer) > 0 and
       separation(Kerb, Observer) < KerbThreshHold
   then
       change-orientation(KerbAvoidanceAngle)
```

- **Planning Layer.** Long-term behavior, e.g. plans trajectories (paths) to goals
- **Modeling layer.** Keeps and modifies environment model; selects new goals for planning layer
- **Control subsystem.** Exceeds control (e.g. by suppressing information input to certain layers („censorship“)

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

Hybrid Architectures
Example Vertical Layering: “InteRRaP”

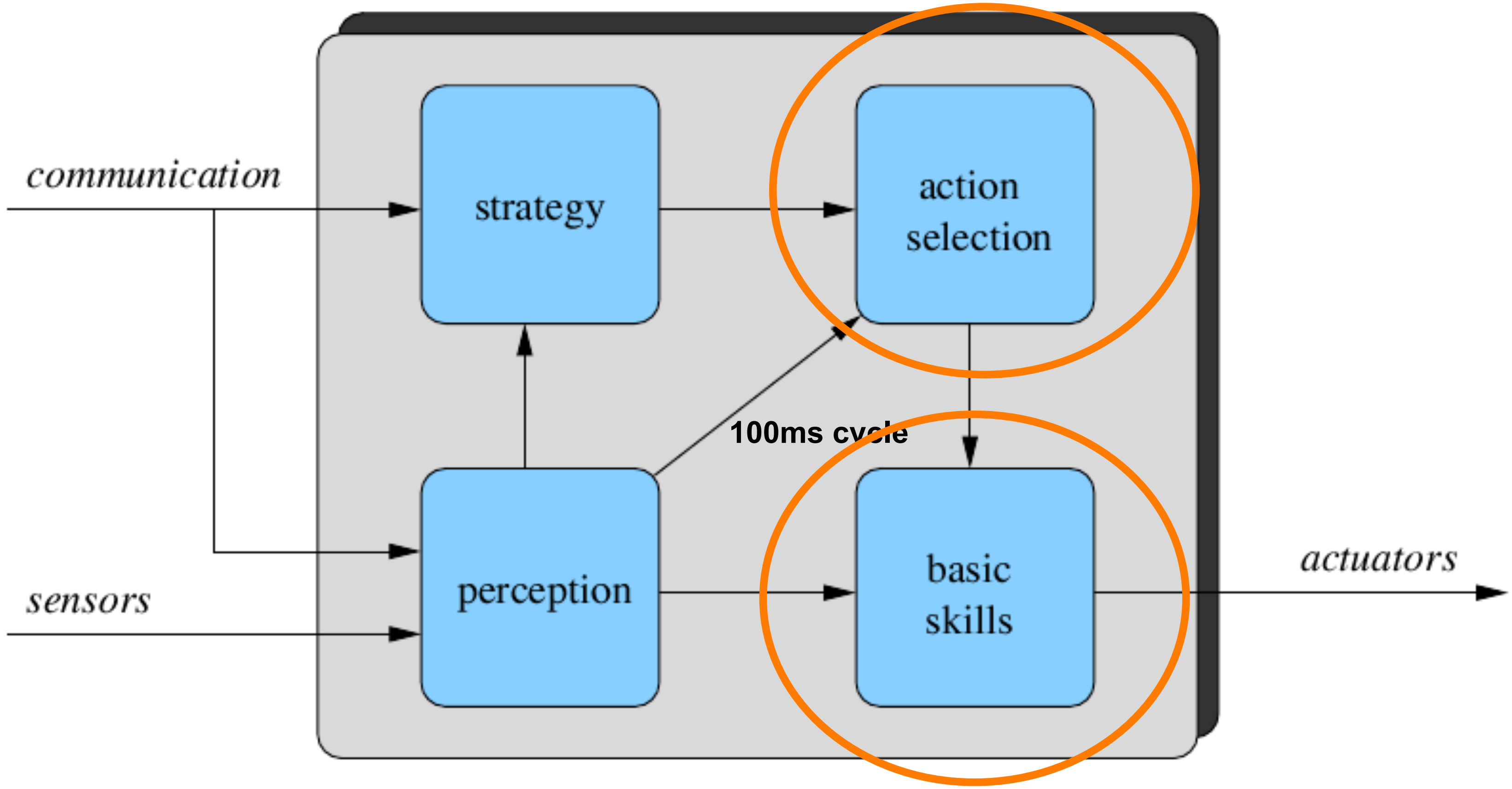
- Bottom-Up-Activation: If lower level layer is not competent for situation → pass control to higher level
- Top-Down-Execution: Higher level layers make use of “facilities” provided by lower level layer



(Mueller 1995)

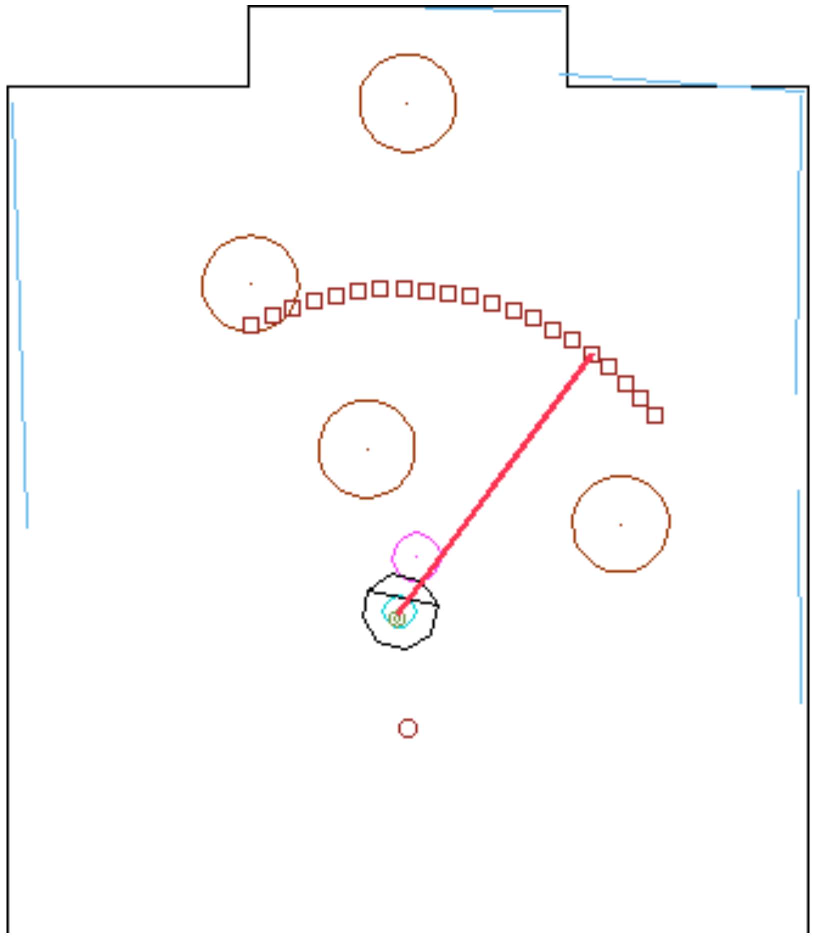
Case study: CS Freiburg Action Selection

Player architecture



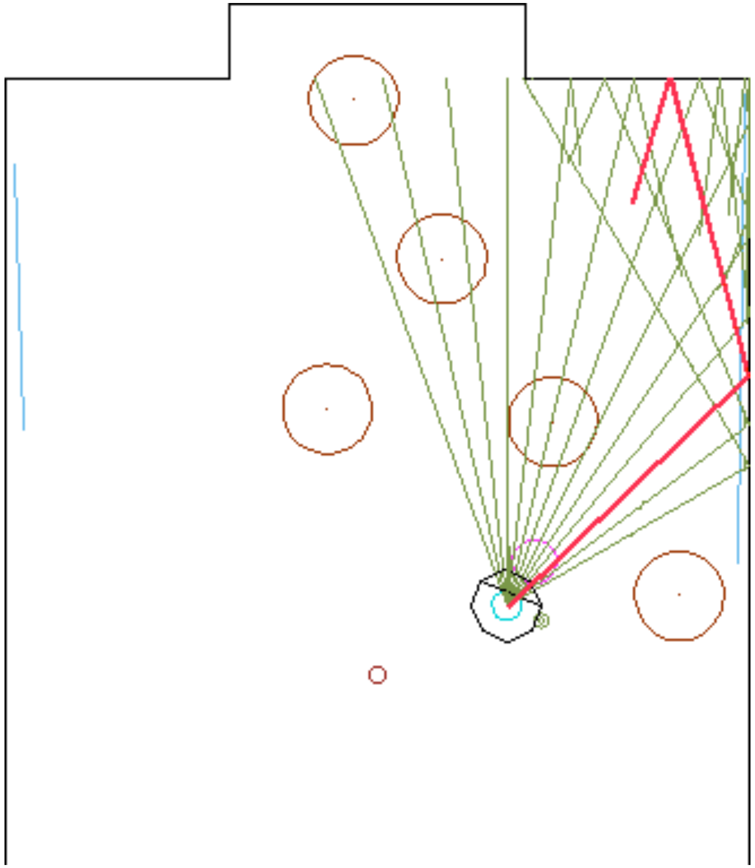
Case study: CS Freiburg Action Selection
Skill example: Dribbling

- Consider points on arc around the robot's location
- Compute utility according to
  - Distance to obstacles (+)
  - Heading angle difference (-)
  - Remaining angle to goal (-)
- Select best angle



Case study: CS Freiburg Action Selection
Skill example: Inbound-shot

- Consider possible shoot directions with predicted reflections
- Compute utility based on
  - Distance to obstacles (-)
  - Heading angle difference (-)
  - Distance to goal at end of line (-)



Case study: CS Freiburg Action Selection

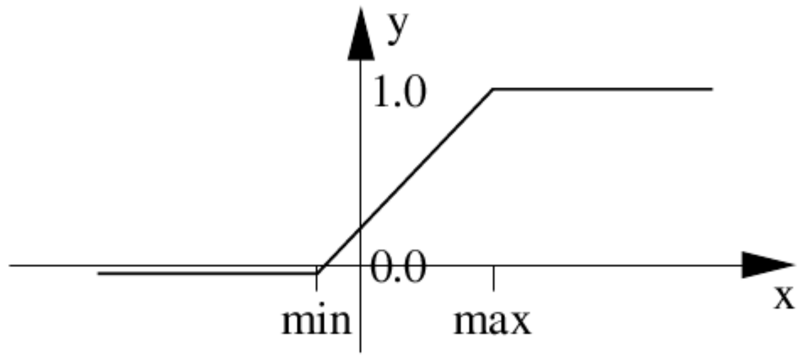
Some basic skills

- The state of the environment, as it is perceived by the agent, is described via a number of continuously valued propositions pi∈[0..1]
- Competence modules are connected with goals if they are able to influence goal conditions and also with each other, if a competence module has an effect that is a precondition of another
- Goals are the source of activation. An action is selected by considering each competence module’s executability and received activation
- The relevance condition *role active* (player has *active* role) ensures that only one of these goals is relevant at a time depending on the player’s current role

Case study: CS Freiburg Action Selection

Propositions

- Are either binary $p \in \{true, false\}$ or continuous $p \in [0..1]$
  - Continuous propositions are generated by simple fuzzification
- Some examples:
  - Ball_present [0,1] true ball position is known



up_straight

```
double StraightUp(double x, double min,
double max)
{
  if(max == min)
      return 0.0;
  if(x < min)
      return 0.0;
  if(x > max)
      return 1.0;
  return((x - min) / (max - min));
}
```

Case study: CS Freiburg Action Selection
Propositions

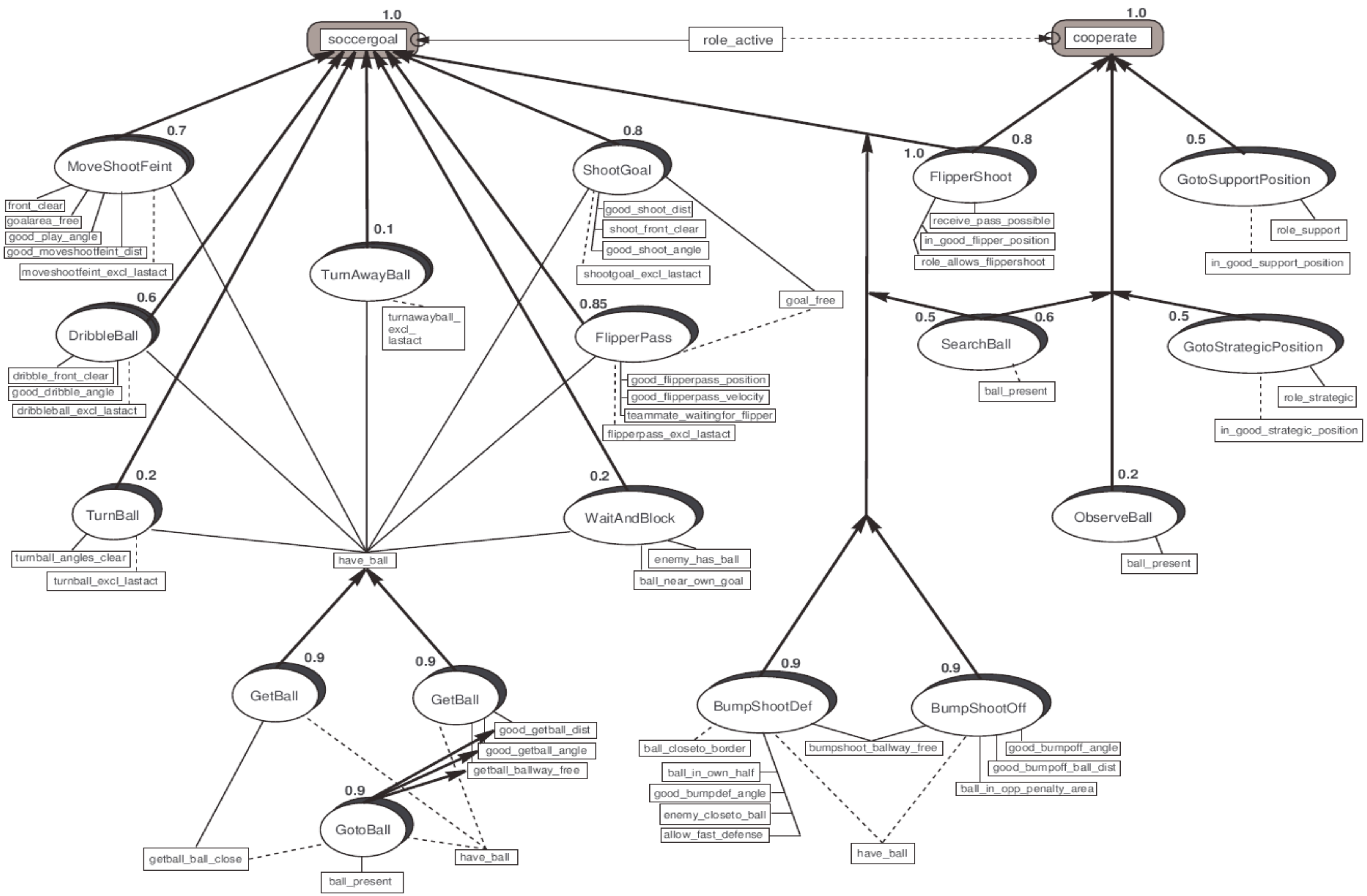
- Only non-conflicting goals; depending on role of player (e.g. active/support):
  - **soccergoal**
  - **cooperate**
- Propositions
  - **ball_present** [0,1] true ball position is known
  - **ball_near_own_goal** as more active as ball is close to goal
  - …
- Reflex behaviors
  - Some simple but important functionality can easier be realized by reactive situation-action rules
    - Robot gets stuck → FreeFromStall
    - 10 seconds rule → GoToPos(FieldCenter)
- Flexibility vs. Persistent
  - Persistence is necessary for successful soccer playing!
  - Achieved by intentionally disallowing undesired action sequences, such as *ShootGoal* → *TribbleBall* (see network slide)

Case Study
CS-Freiburg architecture

- Network has basically non-conflicting goals
  - Goal depends on role of player,
    e.g. support or active
  - Difference to Dorer: Conflicting goal
    "have stamina" and "shoot goal"
- Reflex behaviors
  - Situation-Action rules for urgent situations
    - Robot gets stuck: FreeFromStall
    - 10 seconds rule: GoToPos(FieldCenter)

Case study: CS Freiburg Action Selection

The complete network



Summary

- Logic-based (plan-based) systems can be slow and it is not obvious how to generate symbols from observations
- Behavior-based approaches work bottom up, directly coupling observations to behaviors
- Subsumption architectures are more reactive and more robust, but hard to engineer and it is not obvious how to scale up
- Hybrid architectures combine both approaches
  - behavior-based for low-level/time-critical activities
  - logic-based for high-level mission planning

Literature

- T. Weigel, J.-S. Gutmann, M. Dietl, A. Kleiner and B. Nebel **CS-Freiburg: Coordinating Robots for Successful Soccer Playing** *IEEE Transactions on Robotics and Automation* 18(5):685-699, 2002
- K. Müller **Roboterfußball: Multiagentensystem CS Freiburg**, *Univ. Freiburg*, 2001
- K. Dorer **Behavior Networks for Continuous Domains using Situation-Dependent Motivations** *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, p. 1233-1238, Morgan Kaufmann, Stockholm

www.cs-freiburg.de