Introduction to Multi-Agent Systems

Michal Pechoucek, Branislav Bošanský & Michal Jakob

General Information

Lecturers: Prof. Michal Pěchouček, Dr. Branislav Bošanský, Dr. Michal Jákob

Tutorials: Branislav Bošanský and Karel Horak

14 lectures and 14 tutorials

Course web page:
https://cw.fel.cvut.cz/wiki/courses/be4m36mas/start

Recommended reading:

- J. M. Vidal: Multiagent Systems: with NetLogo Examples (on-line)
- Y. Shoham and K. Leyton-Brown: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations (on-line)
- Russel and Norvig: Artificial Intelligence: Modern Approach

Outline of Lecture 1

1. Motivational Introduction
2. Defining Agency
3. Specifying Agents
4. Agent Architectures

Introduction to Multiagent Systems

Motivational Introduction

Autonomous Agents and Multiagent Systems

Multiagent system is a collection of multiple autonomous agents, each acting towards its objectives while all interacting in a shared environment, being able to communicate and possibly coordinate their actions.

Autonomous agent ~ intelligent agent (see later).

Why Intelligent Agents?

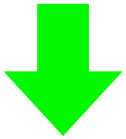
1992: **computers everywhere**

- lots of computerised data
- computer driven manufacturing, production planning, diagnostics

Why Intelligent Agents?

1992: **computers everywhere**

- lots of computerised data
- computer driven manufacturing, production planning, diagnostics



- AI: expert systems, automated planning, machine learning

Why Intelligent Agents?

1992: computers everywhere

Y2K: **internet everywhere**

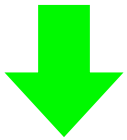
- data provisioning via internet, search (Google from 1998, in 2001 3B of documents)
- an explosion of internet shopping (Amazon from 1995, Ebay from 1996)

Why Intelligent Agents?

1992: computers everywhere

Y2K: **internet everywhere**

- data provisioning via internet, search (Google from 1998, in 2001 3B of documents)
- an explosion of internet shopping (Amazon from 1995, Ebay from 1996)



- parallel computing (map-reduce)
- statistical data analysis and machine learning
- networking, servers

Why Intelligent Agents?

1992: computers everywhere

Y2K: internet everywhere

NOW: **internet of everything**

- mobile computing
- cloud computing
- wireless enabled devices

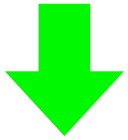
Why Intelligent Agents?

1992: computers everywhere

Y2K: internet everywhere

NOW: **internet of everything**

- mobile computing
- cloud computing
- wireless enabled devices



- Intelligent Agents and Multiagent systems

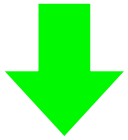
Why Intelligent Agents?

1992: computers everywhere

Y2K: internet everywhere

NOW: **internet of everything**

- mobile computing
- cloud computing
- wireless enabled devices



- Intelligent Agents and Multiagent

Latest trends in computing

Ubiquity: Cost of processing power decreases dramatically (e.g. Moore's Law), computers used everywhere

Interconnection: Formerly only user-computer interaction, nowadays distributed/networked machine-to-machine interactions (e.g. Web APIs)

Complexity: Elaboration of tasks carried out by computers has grown

Delegation: Giving control to computers even in safety-critical tasks (e.g. aircraft or nuclear plant control)

Human-orientation: Increasing use of metaphors that better reflect human intuition from everyday life (e.g. GUIs, speech recognition, object orientation)

Agents briefly

multi-agent system is a decentralized multi-actor (software) system, often geographically distributed whose behavior is defined and implemented by means of complex, peer-to-peer interaction among autonomous, rational and deliberative entities.

autonomous agent is a special kind of a intelligent software program that is capable of highly autonomous rational action, aimed at achieving the private objective of the agent – can exists on its own but often is a component of a multi-agent system – agent is autonomous, reactive, proactive and social

agent researchers study problems of integration, communication, reasoning and knowledge representation, competition (games) and cooperation (robotics), agent oriented software engineering, ...

Agents briefly

agent technology is software technology supporting the development of the autonomous agents and multi-agent systems **agent-based computing** is a special research domain, subfield of computer science and artificial intelligence that studies the concepts of autonomous agents

multi-agent application is a software system, functionality of which is given by interaction among autonomous software/hardware/human components.

- *but also a monolithic software application that is autonomously operating within a community of autonomously acting software applications, hardware systems or human individuals*

Key properties of Intelligent Agent

Autonomy: Agent is fully accountable for its given state. Agent accepts requests from other agents or the environment but decides individually about its actions

Reactivity: Agent is capable of near-real-time decision with respect to changes in the environment or events in its social neighbourhood

Intentionality: Agent maintain long term intention. the agent meets the designer's objectives. It knows its purpose and executes even if not requested.

.

Key properties of Intelligent Agent

Rationality: Agent is capable of intelligent rational decision making. Agent can analyze future course of actions and choose an action which maximizes his utility

Social capability: Agent is aware of the either:

(i) existence,
(ii) communication protocols,
(iii) capability, services provided by the other agents.
Agent can reason about other agents.

Reactivity

- If a program's environment is guaranteed to be fixed, the program need never worry about its own success or failure
- Program just executes blindly.
  - Example of fixed environment: compiler.
- The real world is not like that: most environments are ***dynamic*** and information is ***incomplete***.

Reactivity

- Software is hard to build for dynamic domains: program must take into account possibility of failure
  - ask itself whether it is worth executing!

- A ***reactive*** system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful).

Proactiveness

- Reacting to an environment is easy
  - e.g., stimulus → response rules
- But we generally want agents to ***do things for us***.
  - Hence ***goal directed behaviour***.
- ***Pro-activeness*** = generating and attempting to achieve goals; not driven solely by events; taking the initiative.
  - Also: recognising opportunities.

Social Ability

- The real world is a ***multi***-agent environment: we cannot go around attempting to achieve goals without taking others into account.
  - Some goals can only be achieved by interacting with others.
  - Similarly for many computer environments: witness the INTERNET.

- ***Social ability*** in agents is the ability to interact with other agents (and possibly humans) via ***cooperation***, ***coordination***, and ***negotiation***.
  - At the very least, it means the ability to communicate. . .

Social Ability: Cooperation

- Cooperation is ***working together as a team to achieve a shared goal*.**
- Often prompted either by the fact that no one agent can achieve the goal alone, or that cooperation will obtain a better result (e.g., get result faster).

Social Ability: Coordination

- Coordination is ***managing the interdependencies between activities*.**
- For example, if there is a non-sharable resource that you want to use and I want to use, then we need to coordinate.

Social Ability: Coordination

- Negotiation is the ability to reach ***agreements*** on matters of common interest.

- For example:
  - You have one TV in your house; you want to watch a movie, your housemate wants to watch football.
  - A possible deal: watch football tonight, and a movie tomorrow.

- Typically involves ***offer and counter-offer***, with compromises made by participants.

Some other properties

- Mobility
  - The ability of an agent to move. For software agents this movement is around an electronic network.
- Rationality
  - Whether an agent will act in order to achieve its goals, and will not deliberately act so as to prevent its goals being achieved.
- Veracity
  - Whether an agent will knowingly communicate false information.
- Benevolence
  - Whether agents have conflicting goals, and thus whether they are inherently helpful.
- Learning/adaption
  - Whether agents improve performance over time.

Agents vs. Objects

agent's behaviour is unpredictable as observed from the outside, agent is *situated* in the environment, communication model is *asynchronous*, agent is autonomous, ...

Agents vs. Objects

agent's behaviour is unpredictable as observed from the outside, agent is *situated* in the environment, communication model is *asynchronous,* agent is autonomous, ...

agents are programs, they are build out of objects
$\rightarrow$ while objects often consist of objects, and object make together an object, agents never contain other agents, agents build together a **multiagent system**

Multiagent Systems Engineering &
Agent Oriented Software Engineering

Novel paradigm for building robust, scalable and extensible control, planning and decision-making systems

- *socially-inspired computing*
- *self-organized teamwork systems*
- *distributed (collective) artificial intelligence*

MAS become increasingly relevant as the connectivity, intelligence and autonomy of devices grows!

Software engineering methodology for designing MAS

Multiagent Systems Engineering &
Agent Oriented Software Engineering

Novel paradigm for building robust, scalable and extensible control, planning and decision-making systems

- *socially-inspired computing*
- *self-organized teamwork systems*
- *distributed (collective) artificial intelligence*

MAS become increasingly relevant as the connectivity, intelligence and autonomy of devices grows!

Software engineering methodology for designing MAS

Multiagent Design Problem

Traditional design problem: *How can I build a system that produces the correct output given some input?*

- Each system is more or less isolated, built from scratch

Multiagent Design Problem

Traditional design problem: *How can I build a system that produces the correct output given some input?*

- Each system is more or less isolated, built from scratch

Multiagent design problem: *How can I build a system that can operate independently on my behalf in a networked, distributed, large-scale environment in which it will need to interact with different other components pertaining to other users?*

- Each system is built into an existing, persistent but constantly evolving *computing ecosystem* – it should be robust with respect to changes
- No single owner and/or central authority

Types of Agent Systems

single-agent

multi-agent

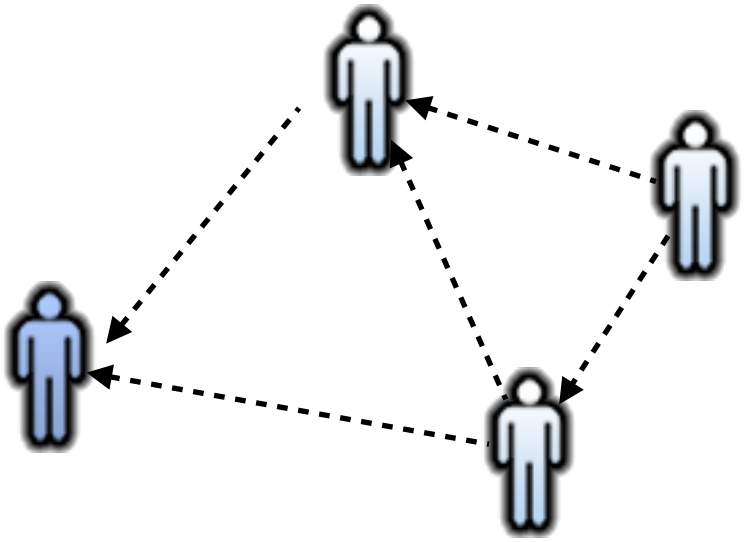
cooperative

single shared utility

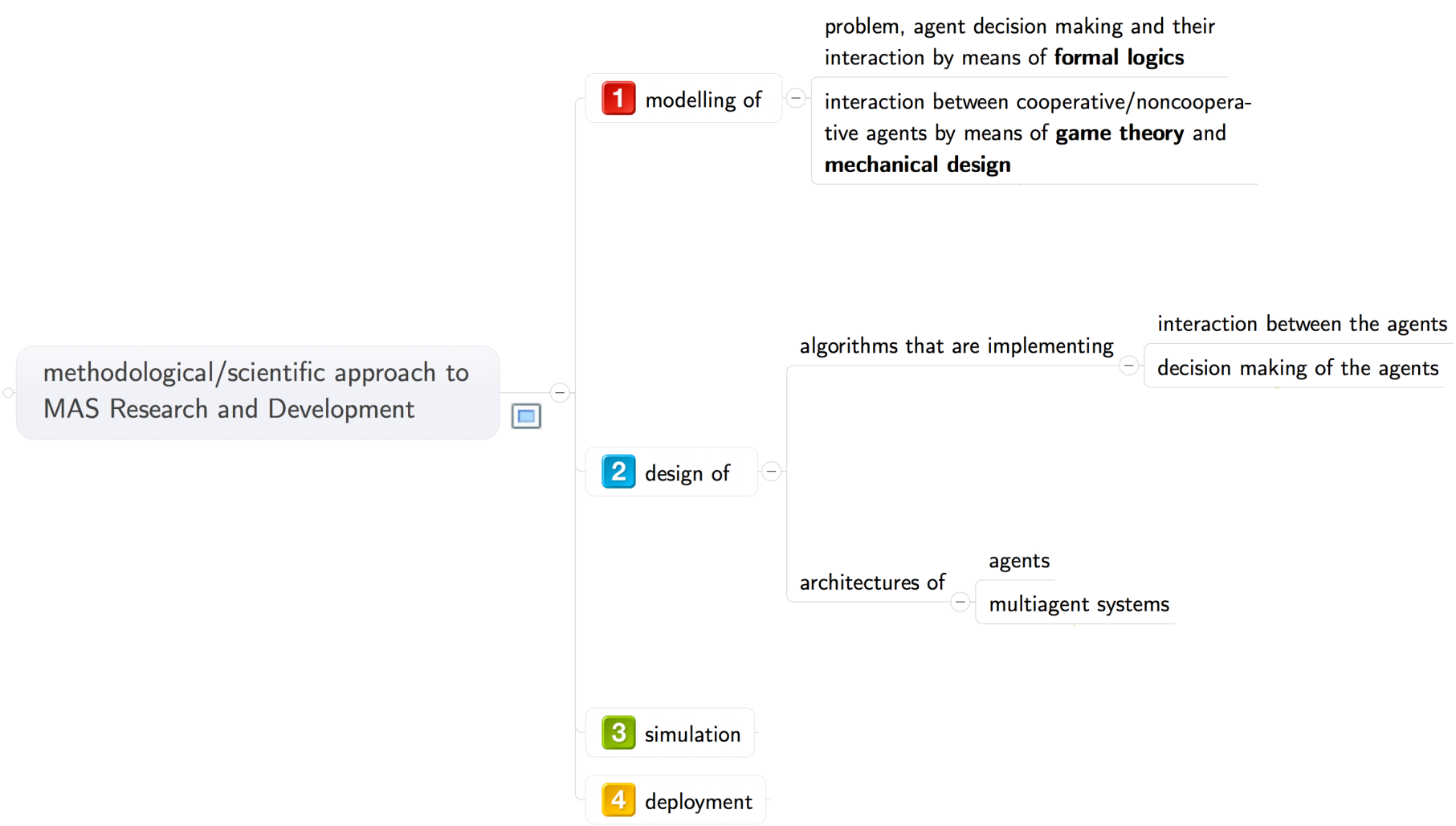
competitive

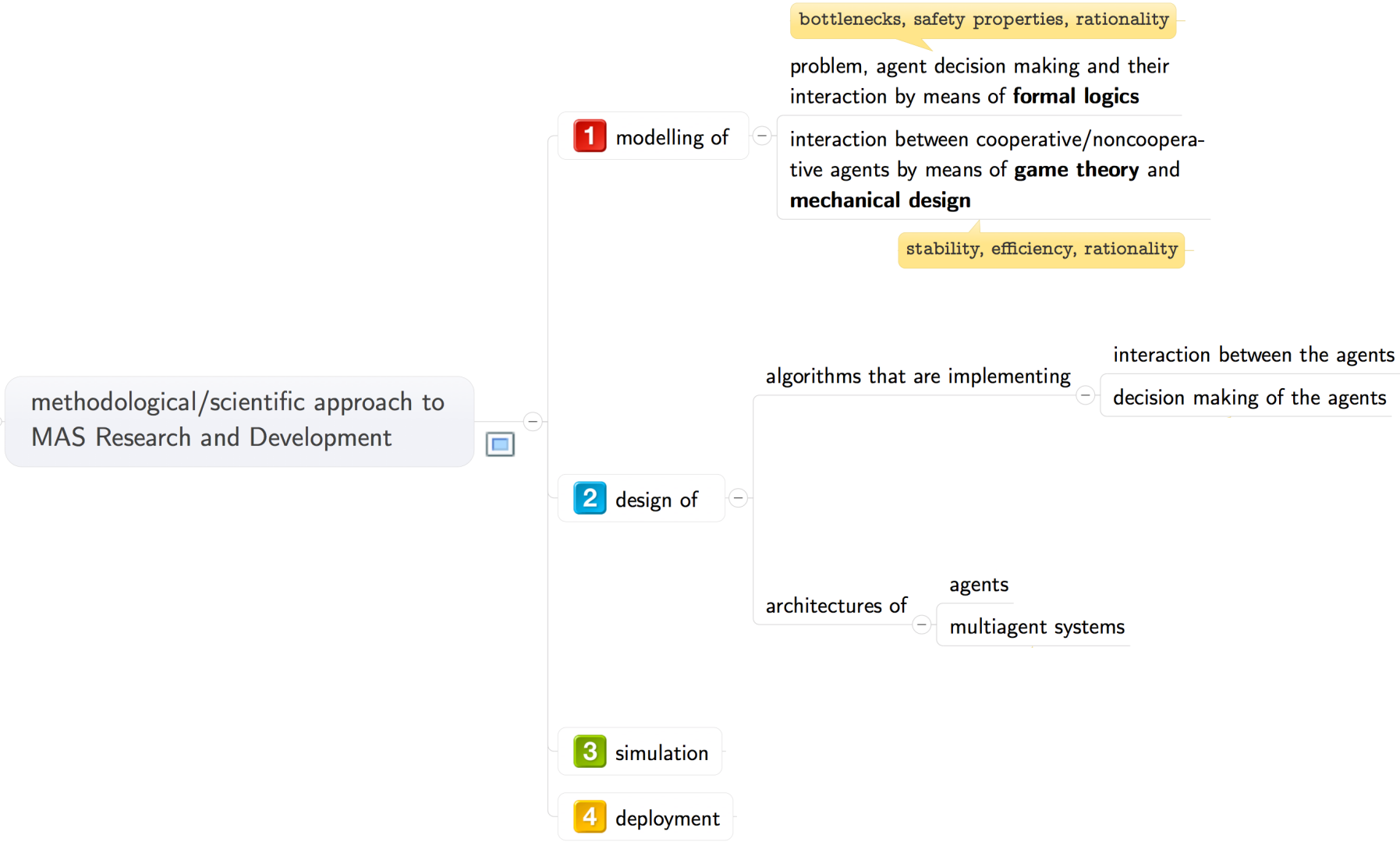
multiple different utilities

Micro vs. Macro MAS Engineering

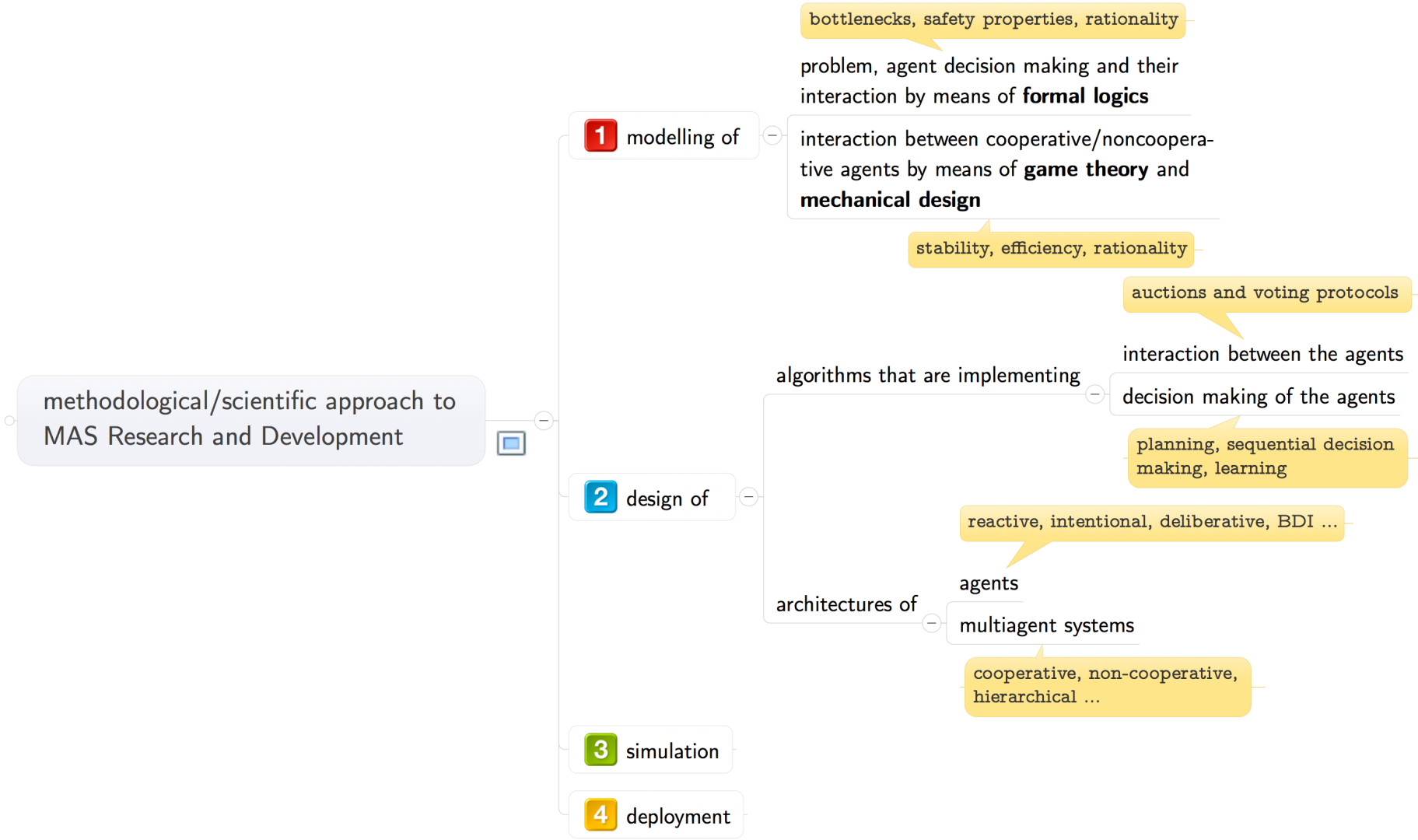


1. The agent design problem (micro perspective):
   How should agents act to carry out their tasks?
2. The society design problem (macro perspective):
   How should agents interact to carry out their tasks?


methodological/scientific approach to MAS Research and Development
1 modelling of
problem, agent decision making and their interaction by means of **formal logics**
interaction between cooperative/noncooperative agents by means of **game theory** and **mechanical design**
2 design of
algorithms that are implementing
interaction between the agents
decision making of the agents
architectures of
agents
multiagent systems
3 simulation
4 deployment


methodological/scientific approach to MAS Research and Development
1 modelling of
problem, agent decision making and their interaction by means of formal logics
bottlenecks, safety properties, rationality
interaction between cooperative/noncooperative agents by means of game theory and mechanical design
stability, efficiency, rationality
2 design of
algorithms that are implementing
interaction between the agents
decision making of the agents
architectures of
agents
multiagent systems
3 simulation
4 deployment


methodological/scientific approach to MAS Research and Development
1 modelling of
problem, agent decision making and their interaction by means of formal logics
bottlenecks, safety properties, rationality
interaction between cooperative/noncooperative agents by means of game theory and mechanical design
stability, efficiency, rationality
2 design of
algorithms that are implementing
interaction between the agents
decision making of the agents
planning, sequential decision making, learning
auctions and voting protocols
architectures of
agents
multiagent systems
3 simulation
4 deployment


methodological/scientific approach to MAS Research and Development
1 modelling of
problem, agent decision making and their interaction by means of formal logics
bottlenecks, safety properties, rationality
interaction between cooperative/noncooperative agents by means of game theory and mechanical design
stability, efficiency, rationality
auctions and voting protocols
2 design of
algorithms that are implementing
interaction between the agents
decision making of the agents
planning, sequential decision making, learning
architectures of
agents
reactive, intentional, deliberative, BDI ...
multiagent systems
cooperative, non-cooperative, hierarchical ...
3 simulation
4 deployment

Opportunities for MAS Deployment

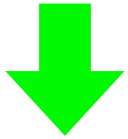
Agent-based computing have been used:

1. **Design paradigm** – the concept of decentralized, interacting, socially aware, autonomous entities as underlying software paradigm (often deployed only in parts, where it suits the application)
2. **Source of technologies** – algorithms, models, techniques architectures, protocols but also software packages that facilitate development of multi-agent systems
3. **Simulation concept** – a specialized software technology that allows simulation of natural multi-agent systems, based on (1) and (2).

Opportunities for MAS Deployment

Agent-based computing have been used:

1. **Design paradigm** – the concept of decentralized, interacting, socially aware, autonomous entities as underlying software paradigm (often deployed only in parts, where it suits the application)

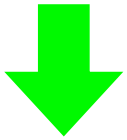


Agent Oriented Software Engineering – provide designers and developers with a way of structuring an application around autonomous, communicative elements, and lead to the construction of software tools and infrastructures to support this metaphor

Opportunities for MAS Deployment

Agent-based computing have been used:

1. **Design paradigm** – the concept of decentralized, interacting, socially aware, autonomous entities as underlying software paradigm (often deployed only in parts, where it suits the application)
2. **Source of technologies** – algorithms, models, techniques architectures, protocols but also software packages that facilitate development of multi-agent systems

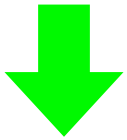


Multi-Agent Techniques – provide a selection of specific computational techniques and algorithms for dealing with collective of computational processes and complexity of interactions in dynamic and open environments.

Opportunities for MAS Deployment

Agent-based computing have been used:

1. **Design paradigm** – the concept of decentralized, interacting, socially aware, autonomous entities as underlying software paradigm (often deployed only in parts, where it suits the application)
2. **Source of technologies** – algorithms, models, techniques architectures, protocols but also software packages that facilitate development of multi-agent systems
3. **Simulation concept** – a specialized software technology that allows simulation of natural multi-agent systems, based on (1) and (2).



Multi-Agent Simulation – provide expressive models for representing complex and dynamic real-world environments, with the emphasis on capturing the interaction related properties of such systems

Intelligent Agents Applications

Manufacturing and production

Traffic and logistics

Robotics, autonomous systems

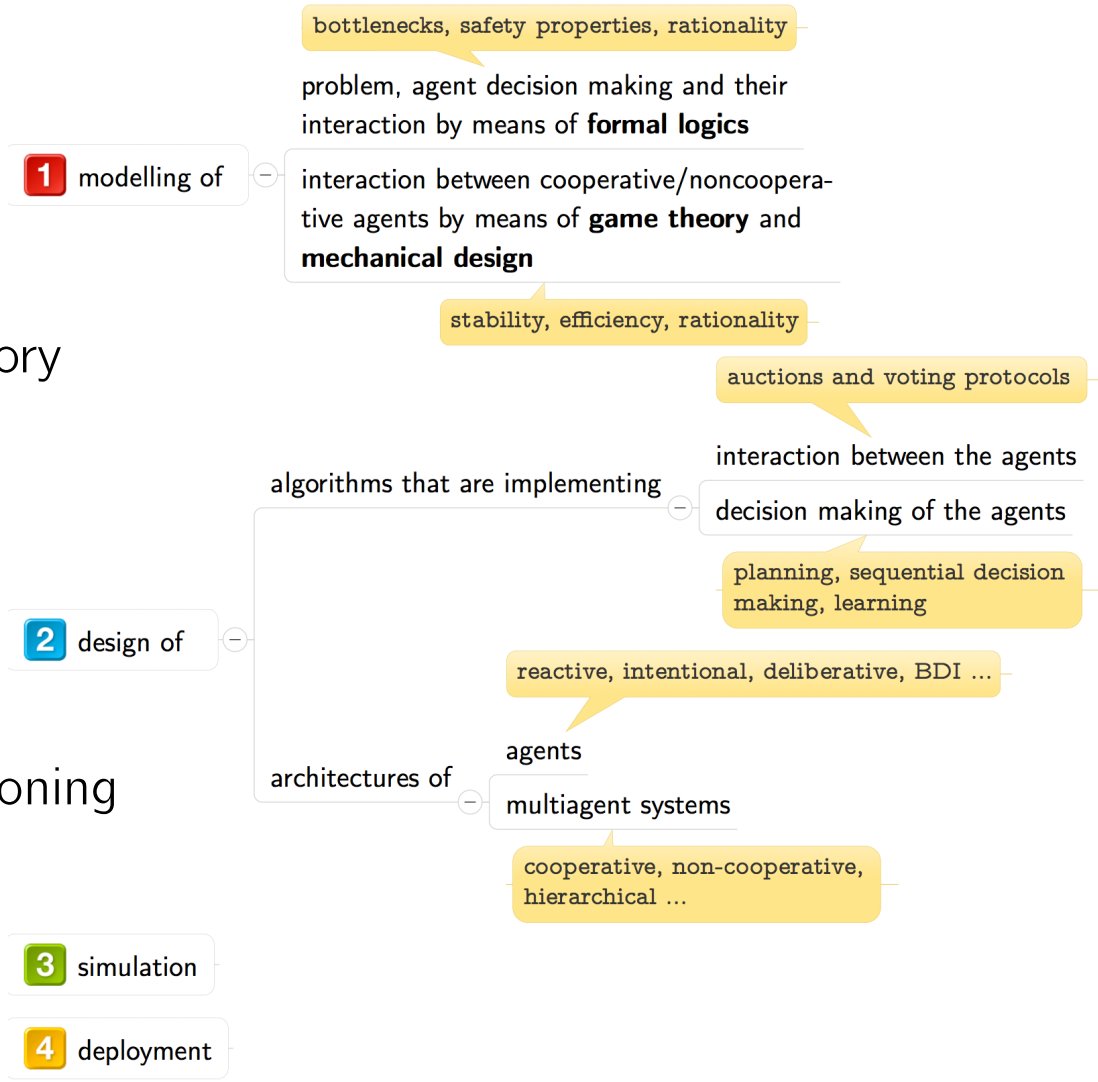
Air traffic and space

Security applications

Energy and smart grids

Course Content

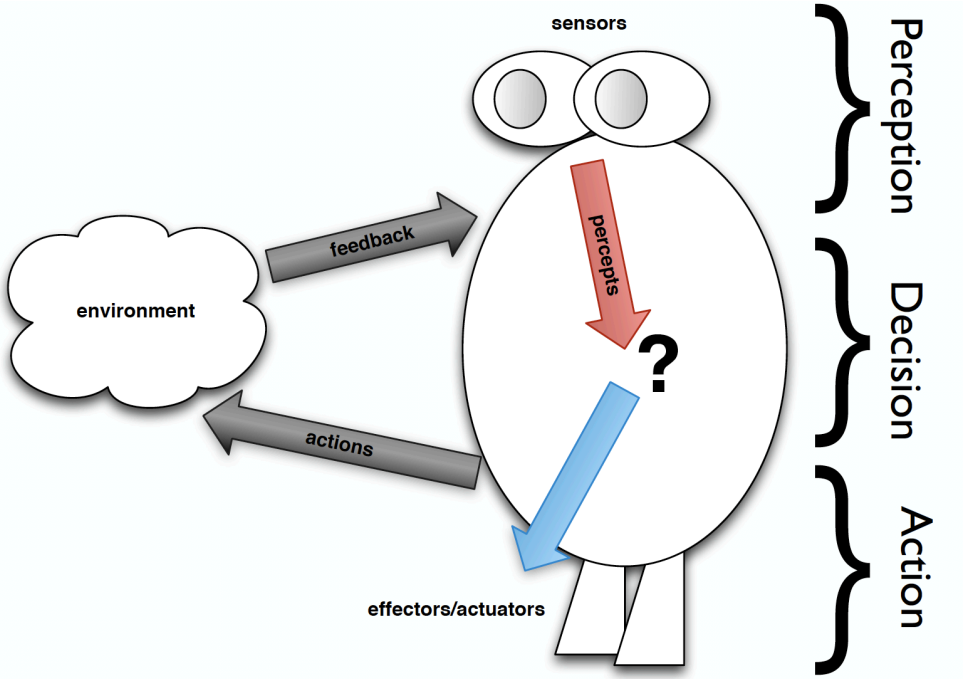
- Agent architectures
- Non-cooperative game theory
- Coalition game theory
- Mechanism design
- Auctions
- Social choice
- Distributed constraint reasoning
- Agent based simulation



Introduction to Multi-Agent Systems

Defining Agency

What is Agent?



Definition (Russell & Norvig): An agent is anything that can perceive its environment (through its sensors) and act upon that environment (through its effectors)

Focus on situatedness in the environment (embodiment)

The agent can only influence the environment but not fully control it (sensor/effector failure, non-determinism)

What is Agent?

Definition (Wooldridge & Jennings): An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives/delegated goals.

Adds a second dimension to agent definition: the relationship between agent and designer/user

- agent is capable of independent action
- agent action is purposeful

Autonomy is a central, distinguishing property of agents

Rational Behaviour

Definition (Russell & Norvig): Rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date and whatever bulit-in knowledge the agent has.

Rationality is relative and depends on four aspects:

1. **performance measure** for the degree of success
2. **percept sequence** (complete perceptual history)
3. **agent's knowledge** about the environment
4. **actions** available to the agent

Agent Environments

- Since agents are in close contact with their environment, the properties of the environment affect agents.
  - Also have a big effect on those of us who build agents.
- Common to categorise environments along some different dimensions.
  - Fully observable vs partially observable
  - Deterministic vs non-deterministic
  - Static vs dynamic
  - Discrete vs continuous
  - Episodic vs non-episodic
  - Real Time

Agent Environments

- Fully observable vs partially observable.
  - An accessible or ***fully observable*** environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state.
  - Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible, or ***partially observable***.
  - The more accessible an environment is, the simpler it is to build agents to operate in it.

Agent Environments

- Fully observable vs partially observable.
  - An accessible or ***fully observable*** environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state.
  - Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible, or ***partially observable***.
  - The more accessible an environment is, the simpler it is to build agents to operate in it.

- Deterministic vs non-deterministic.
  - A deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action.
  - The physical world can to all intents and purposes be regarded as non-deterministic.
  - We'll follow Russell and Norvig in calling environments ***stochastic*** if we quantify the non-determinism using probability theory.
  - Non-deterministic environments present greater problems for the agent designer.

Agent Environments

- Static vs dynamic.
  - A ***static*** environment is one that can be assumed to remain unchanged except by the performance of actions by the agent.
  - A ***dynamic*** environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control.
  - The physical world is a highly dynamic environment.
  - One reason an environment may be dynamic is the presence of other agents.

Agent Environments

- Static vs dynamic.
  - A ***static*** environment is one that can be assumed to remain unchanged except by the performance of actions by the agent.
  - A ***dynamic*** environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control.
  - The physical world is a highly dynamic environment.
  - One reason an environment may be dynamic is the presence of other agents.

- Discrete vs continuous.
  - An environment is discrete if there are a fixed, finite number of actions and percepts in it.
    - Otherwise it is continuous
  - Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.

Agent Environments

- Episodic vs non-episodic
  - In an ***episodic*** environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios.
    - An example of an episodic environment would be an assembly line where an agent had to spot defective parts.
  - Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to perform based only on the current episode — it need not reason about the interactions between this and future episodes.
    - Relations to the Markov property
  - Environments that are not episodic are sometimes called ***non-episodic*** or ***sequential***.
    - Here the current decision affects future decisions.
    - Driving a car is sequential.

Agent Environments

- Real time
  - A ***real time*** interaction is one in which time plays a part in evaluating an agents performance
  - Such interactions include those in which:
    - A decision must be made about some action within a given time bound
    - Some state of affairs must occur as quickly as possible
    - An agent has to repeat some task, with the objective to repeat the task as often as possible

Example Environments

	Solitaire	Backgammon	Shopping	Taxi
Observable				
Deterministic				
Episodic				
Static				
Discrete				
Single-agent				

Rational Behaviour

Definition (Russell & Norvig): Rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date and whatever bulit-in knowledge the agent has.

Rationality is relative and depends on four aspects:

1. **performance measure** for the degree of success
2. **percept sequence** (complete perceptual history)
3. **agent's knowledge** about the environment
4. **actions** available to the agent

Abstract Architectures for Agents

- Assume the world may be in any of a finite set E of discrete, instantaneous states

$$E = \{e, e', \ldots\}$$

- Agents are assumed to have a repertoire of possible actions, Ac, available to them, which transform the state of the world.

$$Ac = \{\alpha, \alpha', \ldots\}$$

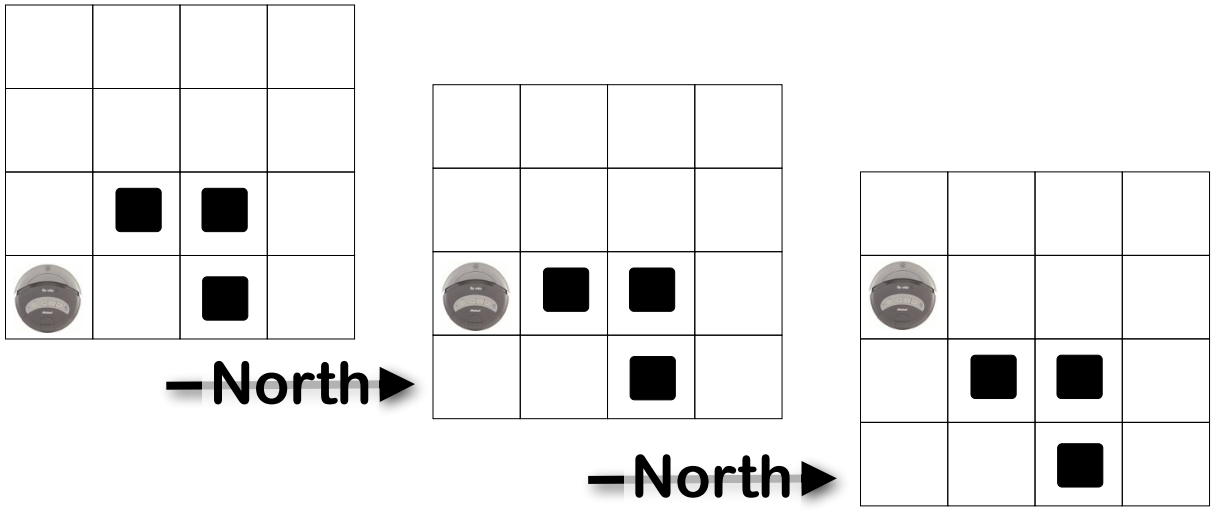
  - Actions can be non-deterministic, but only one state ever results from and action.

- A run, r, of an agent in an environment is a sequence of interleaved world states and actions:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u$$

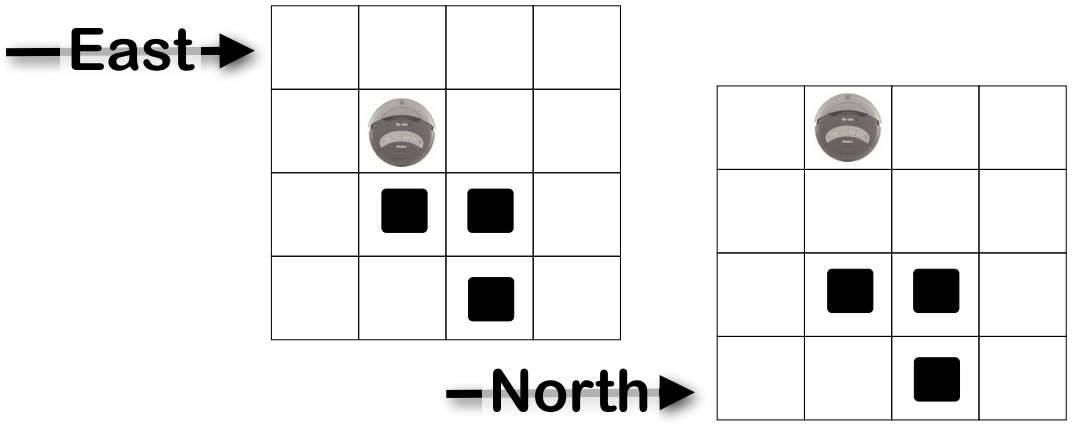
Abstract Architectures for Agents (1)

- When actions are deterministic each state has only one possible successor.
- A run would look something like the following:

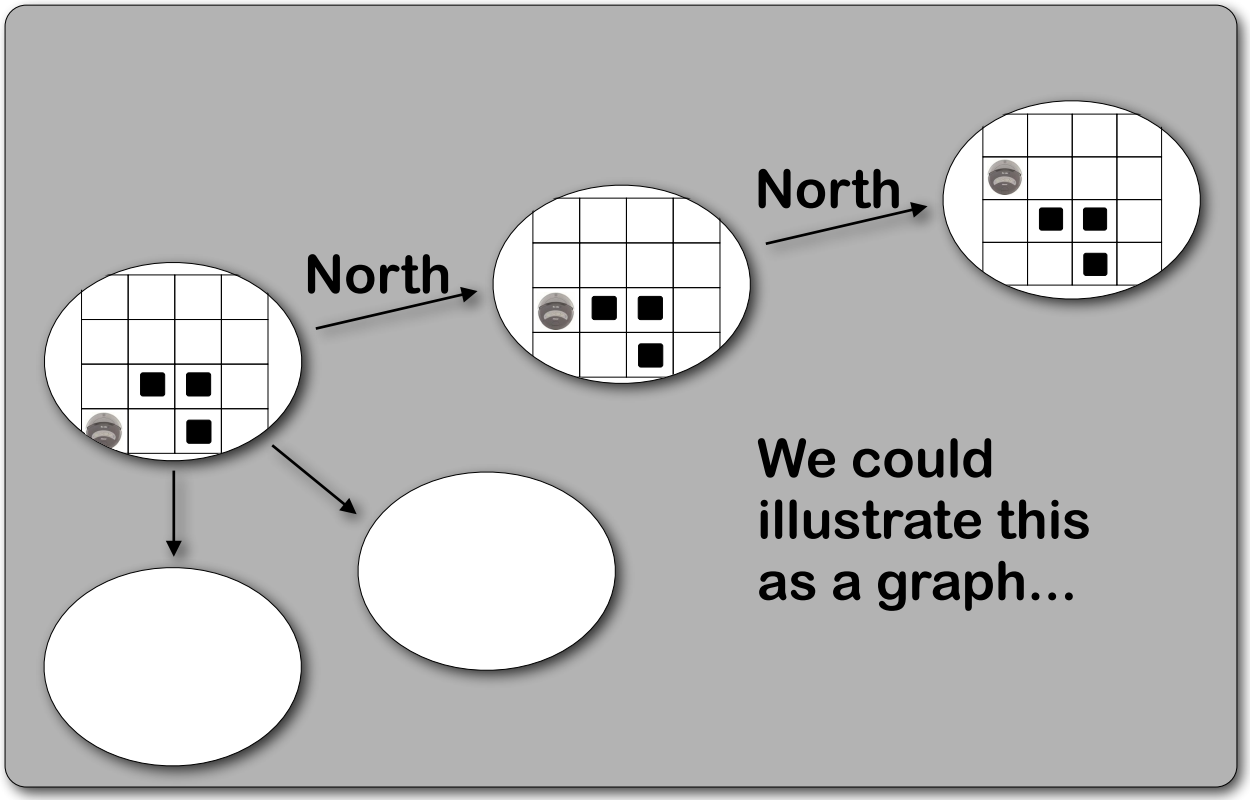


Abstract Architectures for Agents (2)

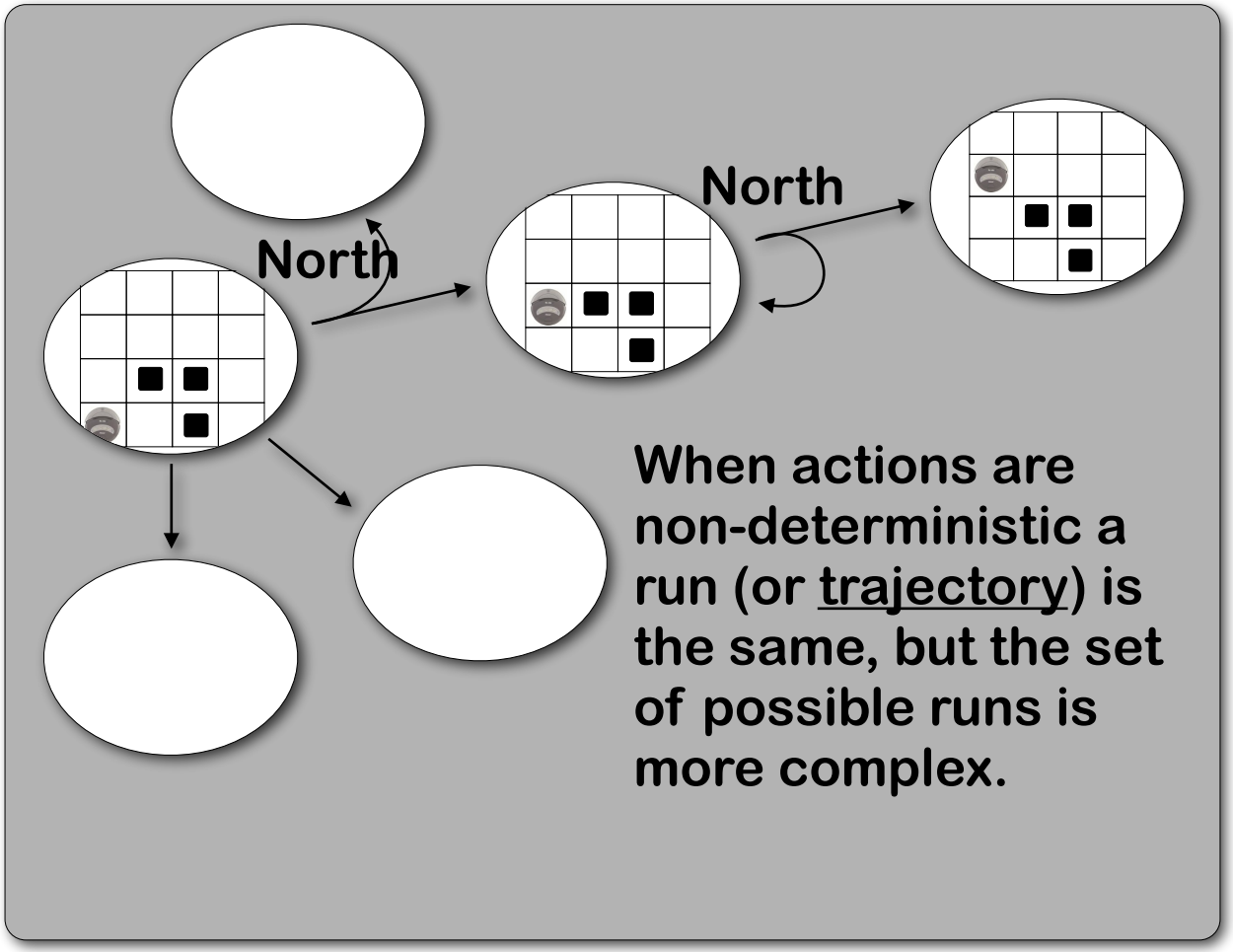
- When actions are deterministic each state has only one possible successor.
- A run would look something like the following:



Abstract Architectures for Agents



Abstract Architectures for Agents



Runs

- In fact it is more complex still, because all of the runs we pictured start from the same state.
- Let:

$\mathcal{R}$ be the set of all such possible finite sequences (over E and Ac);
$\mathcal{R}^{Ac}$ be the subset of these that end with an action; and
$\mathcal{R}^{E}$ be the subset of these that end with a state.

- We will use $r, r', \ldots$ to stand for the members of $\mathcal{R}$
  - These sets of runs contain ***all*** runs from ***all*** starting states.

Environments

- A ***state transformer*** function represents behaviour of the environment:

$$\tau : \mathcal{R}^{Ac} \longrightarrow 2^{E}$$

- Note that environments are...
  - ***history dependent***: the next state not only dependent on the action of the agent, but an earlier action may be significant
  - ***non-deterministic***: There is some uncertainty about the result

- If $\tau(r) = \emptyset$ there are no possible successor states to r, so we say the run has ended. ("Game over.")
  - An environment Env is then a triple $Env = \langle E, e_0, \tau \rangle$ where E is set of states, $e_0 \in E$ is initial state; and τ is state transformer function.

Agents

- We can think of an agent as being a function which maps runs to actions: $Ag : \mathcal{R}^E \rightarrow Ac$
- Thus an agent makes a decision about what action to perform
  - based on the history of the system that it has witnessed to date.
- Let Ag be the set of all agents.

System

- A system is a ***pair*** containing an ***agent*** and an ***environment***.
- Any system will have associated with it a set of possible runs
  - We denote the set of runs of agent Ag in environment Env by:

$$\mathcal{R}(Ag, Env)$$

- Assume that this only contains runs that have ended.

Systems

Formally, a sequence

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)$$

represents a run of an agent Ag in environment $Env = \langle E, e_0, \tau \rangle$ if:

1. e_0 is the initial state of Env
2. $\alpha_0 = Ag(e_0)$; and
3. for $u > 0$,

$$\begin{array}{rcll} e_u & \in & \tau((e_0, \alpha_0, \ldots, \alpha_{u-1})) & \text{and} \\ \alpha_u & = & Ag((e_0, \alpha_0, \ldots, e_u)) & \end{array}$$

Why the notation?

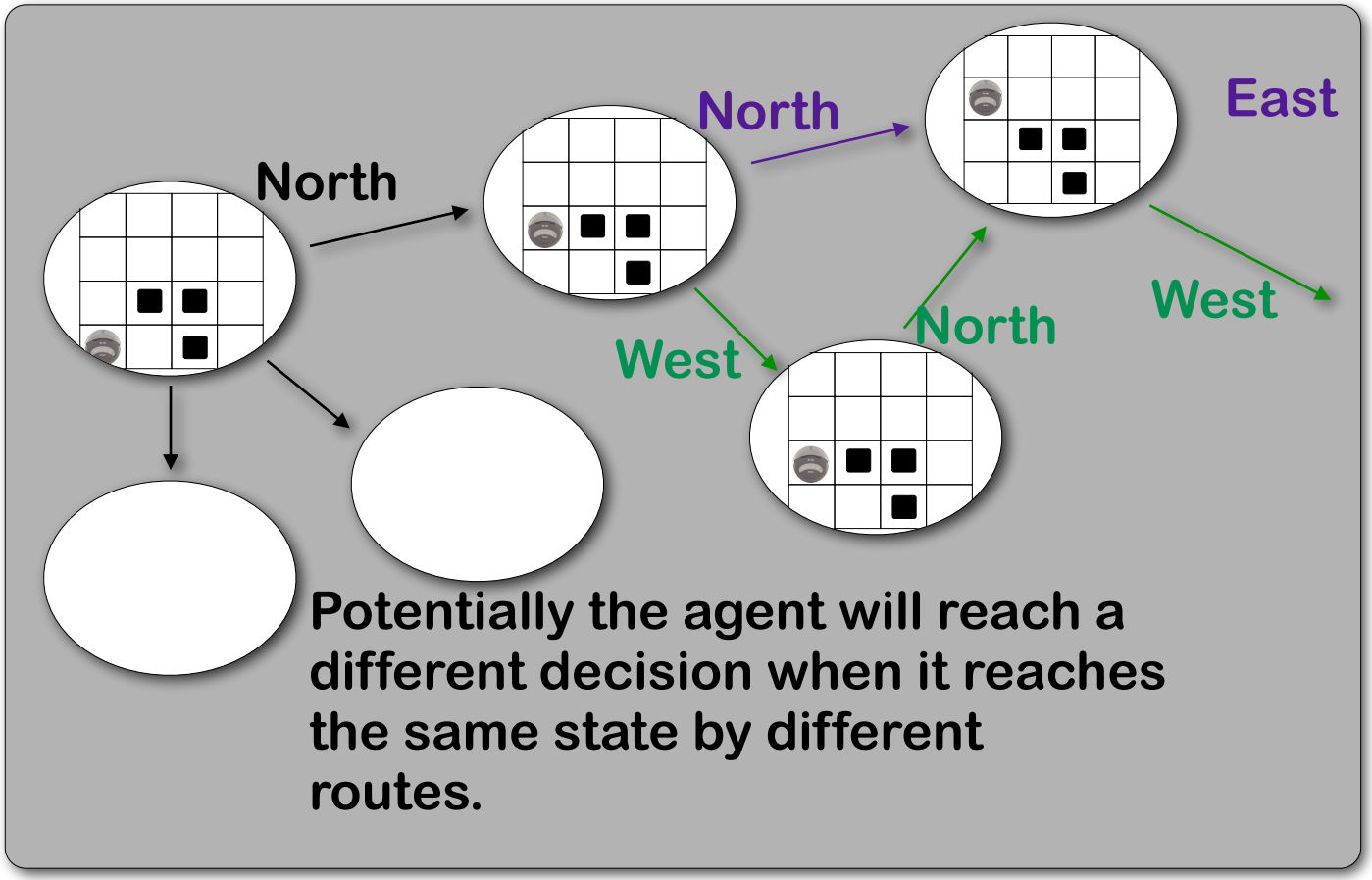
- Well, it allows us to get a precise handle on some ideas about agents.
    - For example, we can tell when two agents are the same.

- Of course, there are different meanings for “same”. Here is one specific one.

> Two agents are said to be *behaviorally equivalent* with respect to Env iff $\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$.

- We won’t be able to tell two such agents apart by watching what they do.

Deliberative Agents



Purely Reactive Agents

- Some agents decide what to do without reference to their history
  - they base their decision making entirely on the present, with no reference at all to the past.

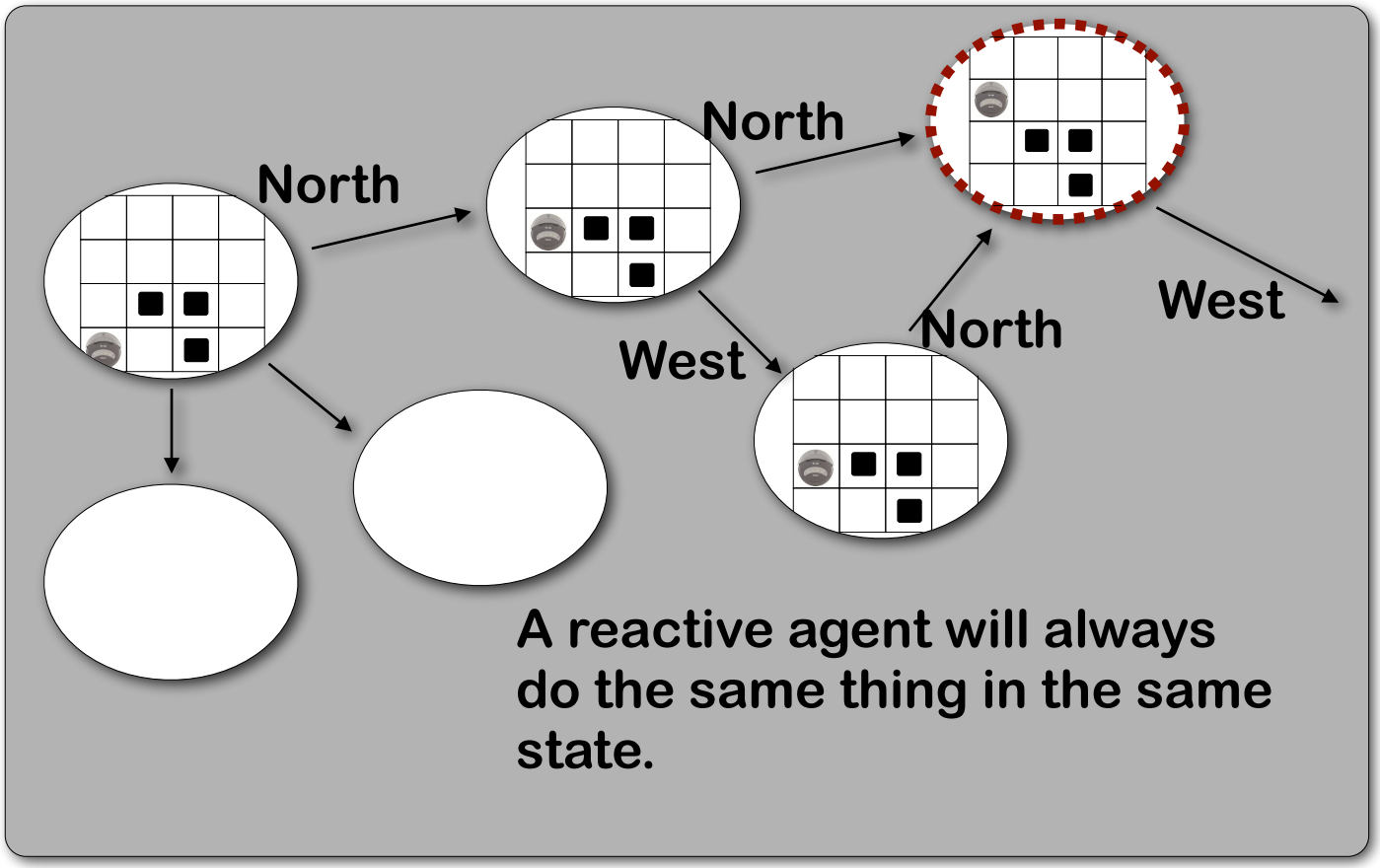
- We call such agents purely reactive:

$$action : E \rightarrow Ac$$

  - A thermostat is a purely reactive agent.

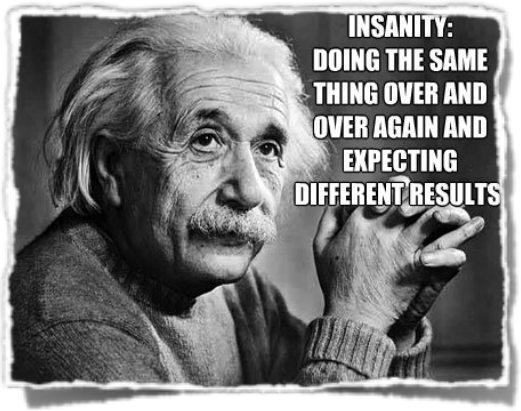
$$action(e) = \begin{cases} \text{off} & \text{if } e = \text{temperature OK} \\ \text{on} & \text{otherwise.} \end{cases}$$

Reactive Agents

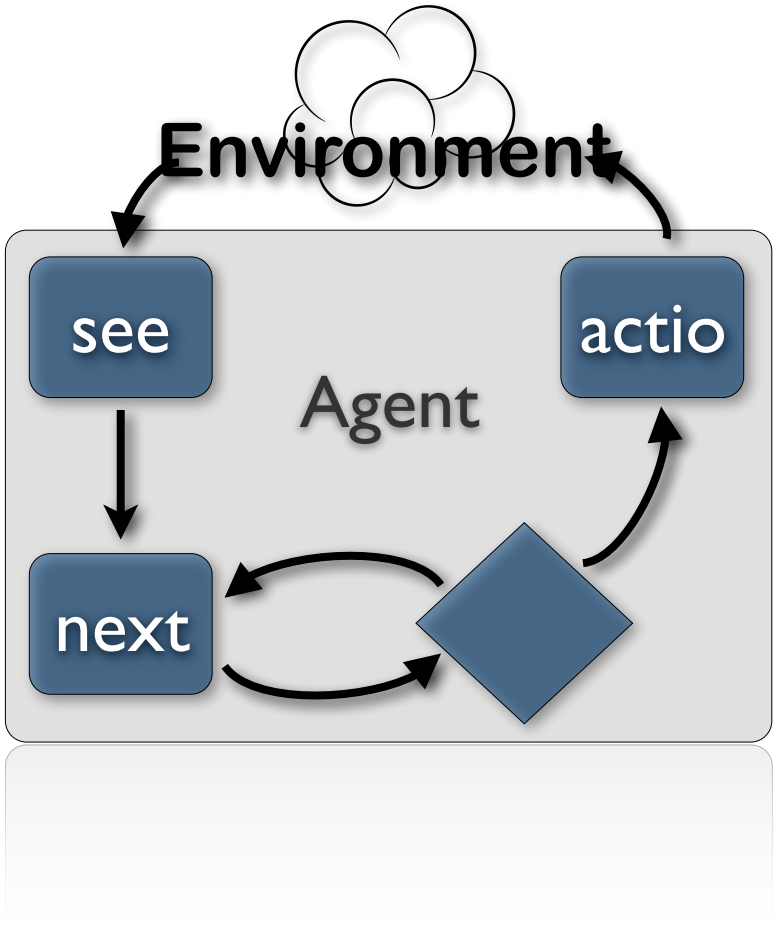


Purely Reactive Robots

- A simple reactive program for a robot might be:
  - *Drive forward until you bump into something. Then, turn to the right. Repeat.*



Agents with State



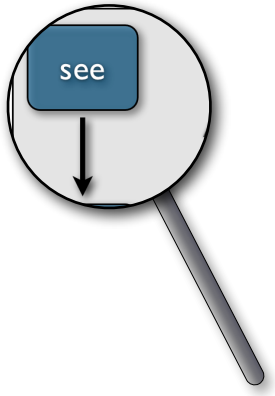
Perception

- The see function is the agent's ability to observe its environment, whereas the action function represents the agent's decision making process.

- Output of the see function is a percept:

$$see : E \rightarrow Per$$

  - ...which maps environment states to percepts.



- The agent has some internal data structure, which is typically used to record information about the environment state and history.

- Let I be the set of all internal states of the agent.

Actions and Next State Functions

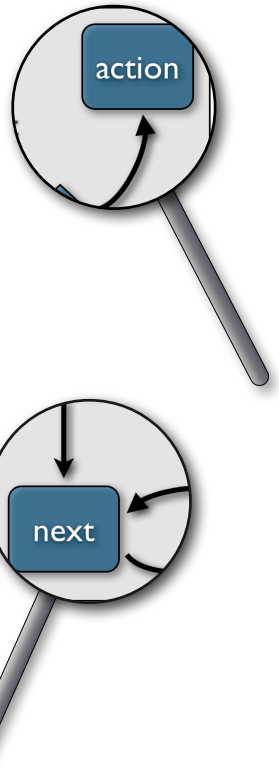
- The action-selection function $action$ is now defined as a mapping from internal states to actions:

$$action : I \rightarrow Ac$$

- An additional function $next$ is introduced, which maps an internal state and percept to an internal state:

$$next : I \times Per \rightarrow I$$

- This says how the agent updates its view of the world ***when it gets a new percept***.



Agent Control Loop

1. Agent starts in some initial internal state i_0.
2. Observes its environment state e, and generates a percept $see(e)$.
3. Internal state of the agent is then updated via $next$ function, becoming $next(i_0, see(e))$.
4. The action selected by the agent is $action(next(i_0, see(e)))$.

   This action is then performed.
5. Goto (2).

Tasks for Agents

- We build agents in order to carry out ***tasks*** for us.
  - The task must be ***specified*** by us. . .
- But we want to tell agents what to do ***without*** telling them how to do it.
  - How can we make this happen???

Utility functions

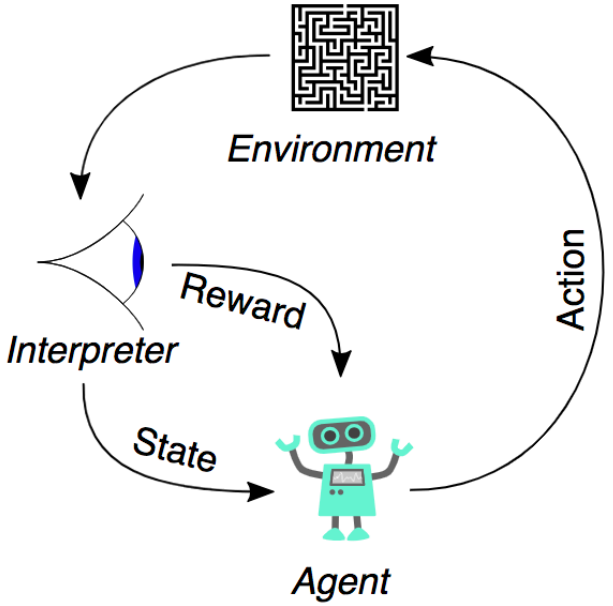
- One idea:
  - associated ***rewards*** with states that we want agents to bring about.
  - We associate ***utilities*** with individual states
    - the task of the agent is then to bring about states that maximise utility.

- A ***task specification*** is then a function which associates a real number with every environment state:

$$u : E \rightarrow \mathbb{R}$$

Local Utility Functions

- But what is the value of a run...
  - minimum utility of state on run?
  - maximum utility of state on run?
  - sum of utilities of states on run?
  - average?



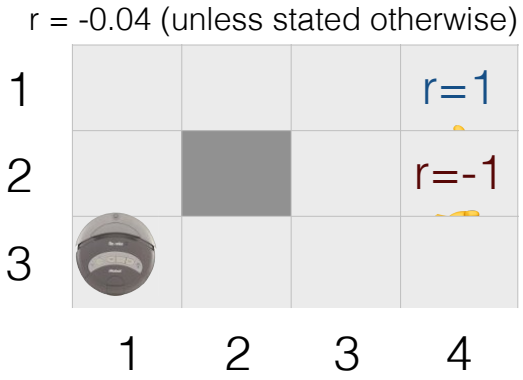
- Disadvantage:
  - difficult to specify a ***long term*** view when assigning utilities to individual states.

- One possibility:
  - a ***discount*** for states later on. This is what we do in ***reinforcement learning***.

Example of local utility function

- Goal is to select actions to ***maximise future rewards***
  - Each action results in moving to a state with some assigned reward
  - Allocation of that reward may be immediate or delayed (e.g. until the end of the run)
  - It may be better to sacrifice immediate reward to gain more long-term reward
- We can illustrate with a simple 4x3 environment
  - What actions maximise the reward?

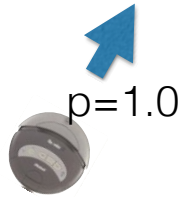
Example of local utility function



Assume environment was ***deterministic***

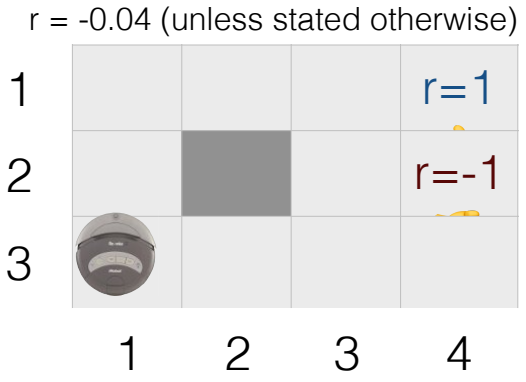
- ***Optimal Solution*** is:
  - [Up, Up, Right, Right, Right]
- ***Additive Reward*** is:
  - r = (-0.04 x 4) + 1.0
  - r = 1.0 - 0.16 = 0.84

Deterministic Environment
Agent is guaranteed to be in the intended cell (i.e. probability = 1.0)



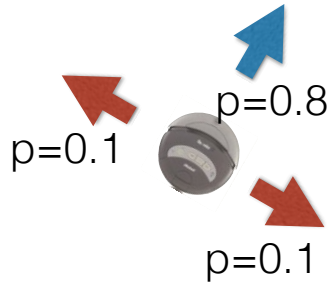
- i.e. the utility gained is the sum of the rewards received
  - The negative (-0.04) reward incentivises the agent to reach its goal asap.

Sequential Decision Making



When environment is ***non-deterministic***

- Probability of reaching the goal if successful:
  - $p = 0.8^5 = 0.32768$
- Could also reach the goal accidentally by going the wrong way round:
  - $p = 0.1^4 \times 0.8 = 0.0001 \times 0.8 = 0.00008$
  - Final probability of reaching the goal: $p = 0.32776$
- Utility gained depends on the route taken
  - Reinforcement Learning builds upon this type of model



Non-Deterministic Environment
Agent may fail to reach its intended cell (i.e. probability of success = 0.8, but may move sideways with p=0.1 in each direction

Utilities over Runs

- Another possibility: assigns a utility not to individual states, but to runs themselves:

$$u : \mathcal{R} \rightarrow \mathbb{R}$$

- Such an approach takes an inherently ***long term view***.

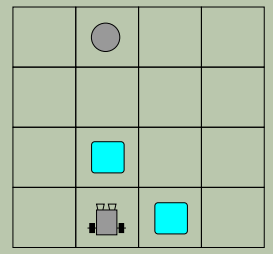
- Other variations:
  - incorporate probabilities of different states emerging.

- To see where utilities might come from, let's look at an example.

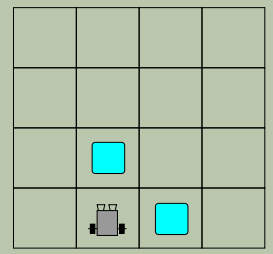
Utility in the Tileworld

- Simulated two dimensional grid environment on which there are ***agents***, ***tiles***, ***obstacles***, and ***holes***.
- An agent can move in four directions:
  - up, down, left, or right
  - If it is located next to a tile, it can push it.
- Holes have to be filled up with tiles.
  - An agent scores points by filling holes with tiles, with the aim to fill as many holes as possible.
- TILEWORLD changes with the random appearance and disappearance of holes.

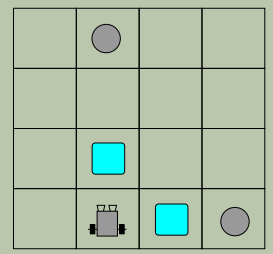
The agent starts to push a tile towards the hole.



But then the hole disappears!!!



Later, a much more convenient hole appears (bottom right)



Utilities in the Tileworld

- Utilities are associated over runs, so that more holes filled is a higher utility.
  - Utility function defined as follows:
  - Thus:

$$u(r) \hat{=} \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

    - if agent fills all holes, utility = 1.
    - if agent fills no holes, utility = 0.

- TILEWORLD captures the need for ***reactivity*** and for the advantages of exploiting opportunities.

Expected Utility

- To denote probability that run r occurs when agent Ag is placed in environment Env, we can write:

$$P(r \mid Ag, Env)$$

- In a non-deterministic environment, for example, this can be computed from the probability of each step.

For a run $r = (e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)$:

$$P(r \mid Ag, Env) = P(e_1, \mid e_0, \alpha_0) P(e_2 \mid e_1, \alpha_1) \ldots$$

and clearly:

$$\sum_{r \in \mathcal{R}(Ag, Env)} P(r \mid Ag, Env) = 1.$$

Expected Utility

- The expected utility (EU) of agent Ag in environment Env (given P, u), is then:

$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r \mid Ag, Env).$$

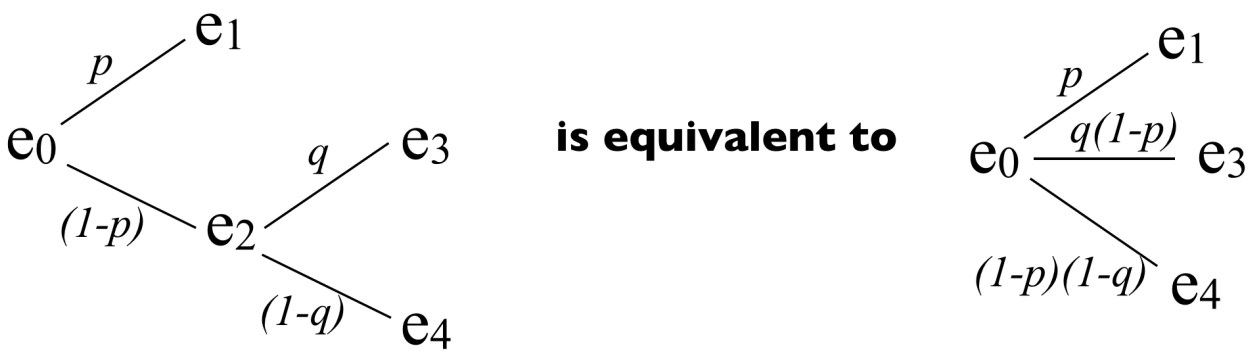
- That is, for each run we compute the utility and multiply it by the probability of the run.

- The expected utility is then the sum of all of these.

Expected Utility

- The probability of a run can be determined from individual actions within a run
  - Using the decomposability axiom from Utility Theory

> *"... Compound lotteries can be reduced to simpler ones using the law of probability. Known as the "no*



Optimal Agents

- The optimal agent Ag_{opt} in an environment Env is the one that maximizes expected utility:

$$Ag_{opt} = \arg \max_{Ag \in \mathcal{AG}} EU(Ag, Env)$$

- Of course, the fact that an agent is optimal does ***not mean that it will*** be best; only that ***on average***, we can expect it to do best.

Example 1

Consider the environment $Env_1 = \langle E, e_0, \tau \rangle$ defined as follows:

$$E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$$

$$\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2\}$$

$$\tau(e_0 \xrightarrow{\alpha_1}) = \{e_3, e_4, e_5\}$$

There are two agents possible with respect to this environment:

$$Ag_1(e_0) = \alpha_0$$

$$Ag_2(e_0) = \alpha_1$$

The probabilities of the various runs are as follows:

$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_1, Env_1) = 0.4$$

$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_1, Env_1) = 0.6$$

$$P(e_0 \xrightarrow{\alpha_1} e_3 \mid Ag_2, Env_1) = 0.1$$

$$P(e_0 \xrightarrow{\alpha_1} e_4 \mid Ag_2, Env_1) = 0.2$$

$$P(e_0 \xrightarrow{\alpha_1} e_5 \mid Ag_2, Env_1) = 0.7$$

Assume the utility function u_1 is defined as follows:

$$u_1(e_0 \xrightarrow{\alpha_0} e_1) = 8$$

$$u_1(e_0 \xrightarrow{\alpha_0} e_2) = 11$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_3) = 70$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_4) = 9$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_5) = 10$$

What are the expected utilities of the agents for this utility function?

Example 1 Solution

Given the utility function u_1 in the question, we have two transition functions defined as $\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2, e_3\}$, and $\tau(e_0 \xrightarrow{\alpha_1}) = \{e_4, e_5, e_6\}$. The probabilities of the various runs (two for the first agent and three for the second) is given in the question, along with the probability of each run occurring. Given the definition of the utility function u_1, the *expected utilities* of agents Ag_0 and Ag_1 in environment Env can be calculated using:

$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r | Ag, Env).$$

This is equivalent to calculating the sum of the product of each utility for a run ending in some state with the probability of performing that run; i.e.

- Utility of $Ag_0 = (0.4 \times 8) + (0.6 \times 11) = 9.8$
- Utility of $Ag_1 = (0.1 \times 70) + (0.2 \times 9) + (0.7 \times 10) = 15.8$

Therefore agent Ag_1 is optimal.

Example 2

Consider the environment $Env_1 = \langle E, e_0, \tau \rangle$ defined as follows:

$$E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$$

$$\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2\}$$

$$\tau(e_1 \xrightarrow{\alpha_1}) = \{e_3\}$$

$$\tau(e_2 \xrightarrow{\alpha_2}) = \{e_4, e_5\}$$

There are two agents, Ag_1 and Ag_2, with respect to this environment:

$$Ag_1(e_0) = \alpha_0 \quad Ag_1(e_1) = \alpha_1 \qquad Ag_2(e_0) = \alpha_0 \quad Ag_2(e_2) = \alpha_2$$

The probabilities of the various runs are as follows:

$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_1, Env_1) = 0.5$$

$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_1, Env_1) = 0.5$$

$$P(e_1 \xrightarrow{\alpha_1} e_3 \mid Ag_1, Env_1) = 1.0$$

$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_2, Env_1) = 0.1$$

$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_2, Env_1) = 0.9$$

$$P(e_2 \xrightarrow{\alpha_2} e_4 \mid Ag_2, Env_1) = 0.4$$

$$P(e_2 \xrightarrow{\alpha_2} e_5 \mid Ag_2, Env_1) = 0.6$$

Assume the utility function u_1 is defined as follows:

$$u_1(e_0 \xrightarrow{\alpha_0} e_1) = 4$$

$$u_1(e_0 \xrightarrow{\alpha_0} e_2) = 3$$

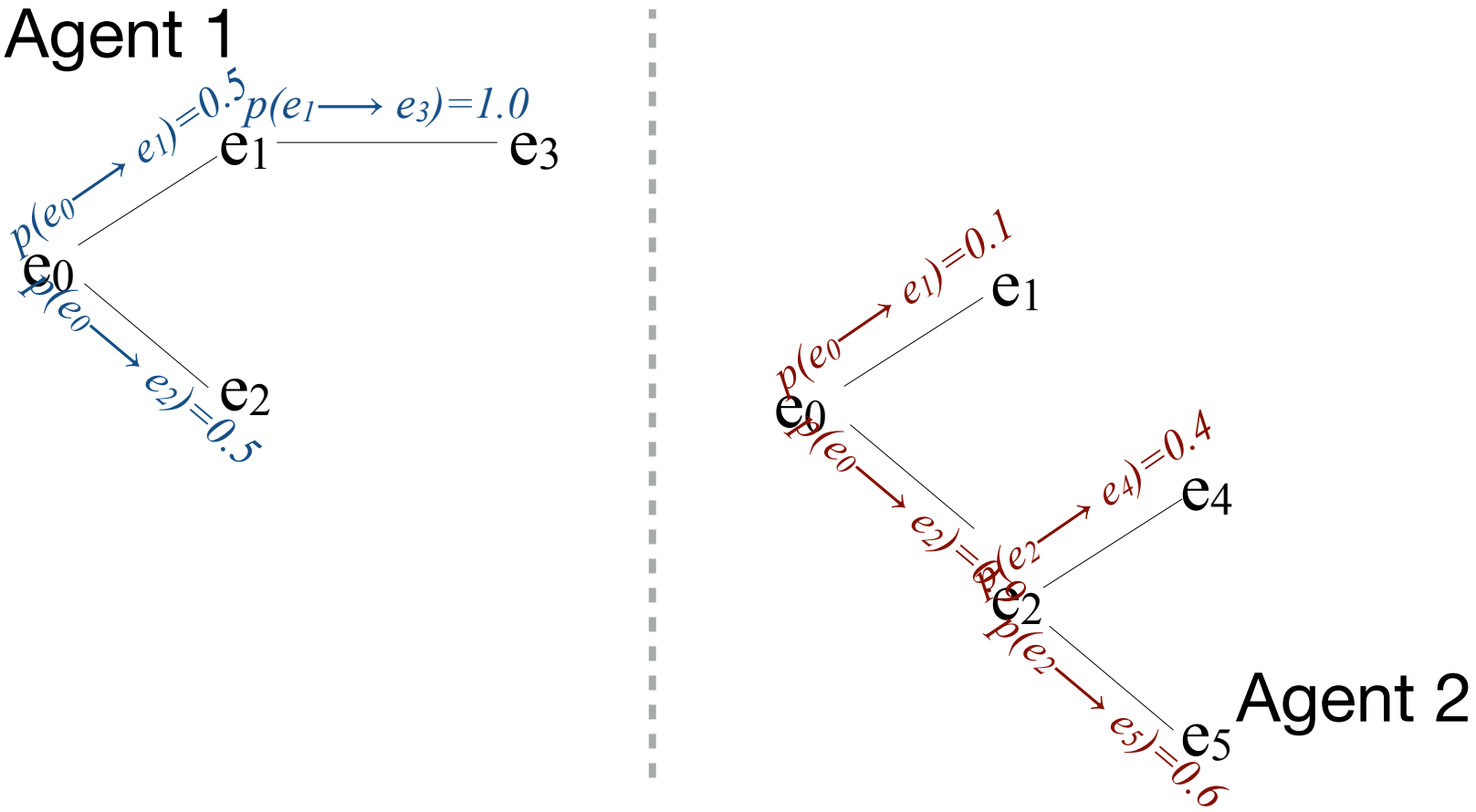
$$u_1(e_1 \xrightarrow{\alpha_1} e_3) = 7$$

$$u_1(e_2 \xrightarrow{\alpha_2} e_4) = 3$$

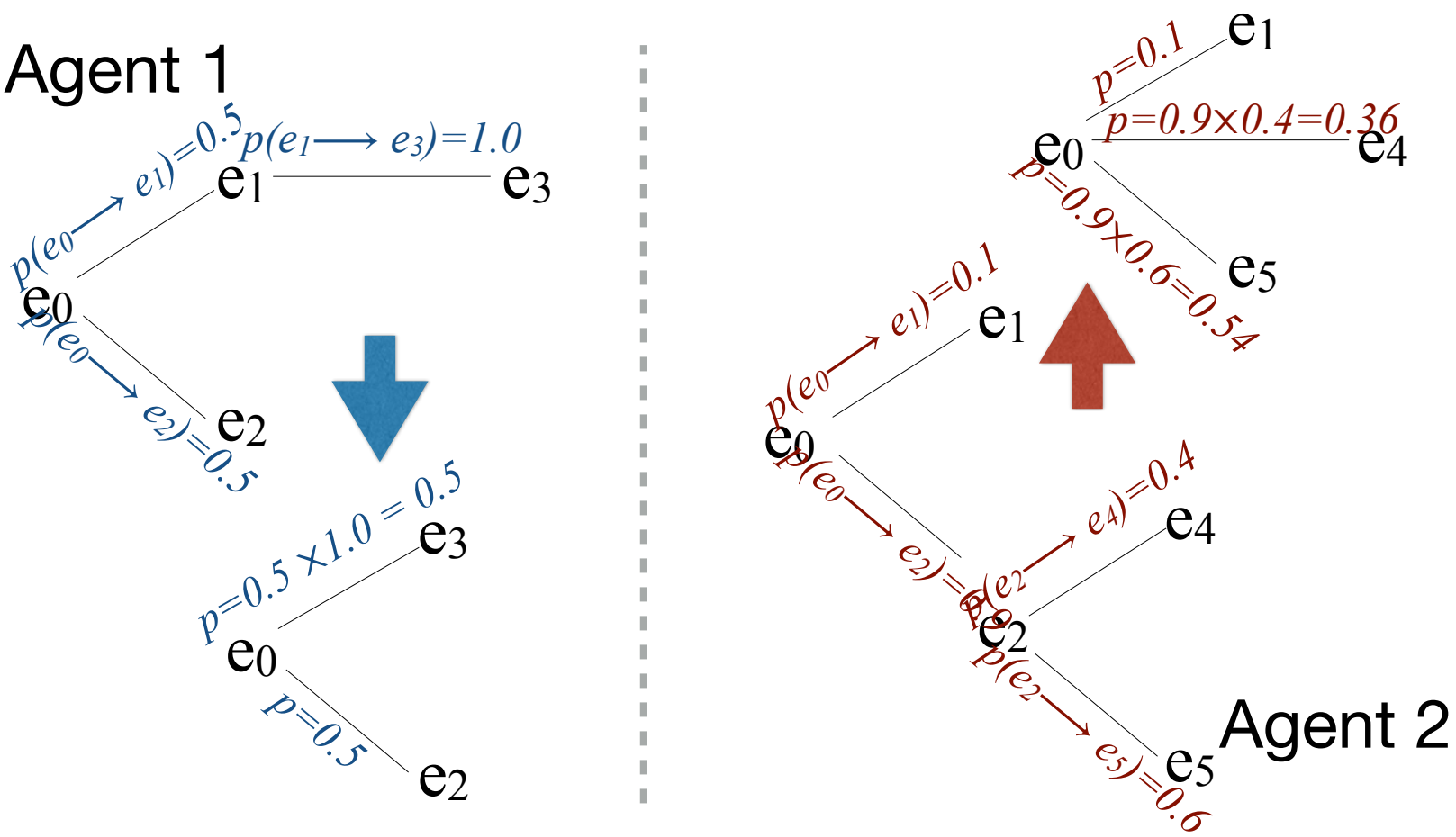
$$u_1(e_2 \xrightarrow{\alpha_2} e_5) = 2$$

What are the expected utilities of the agents for this utility function?

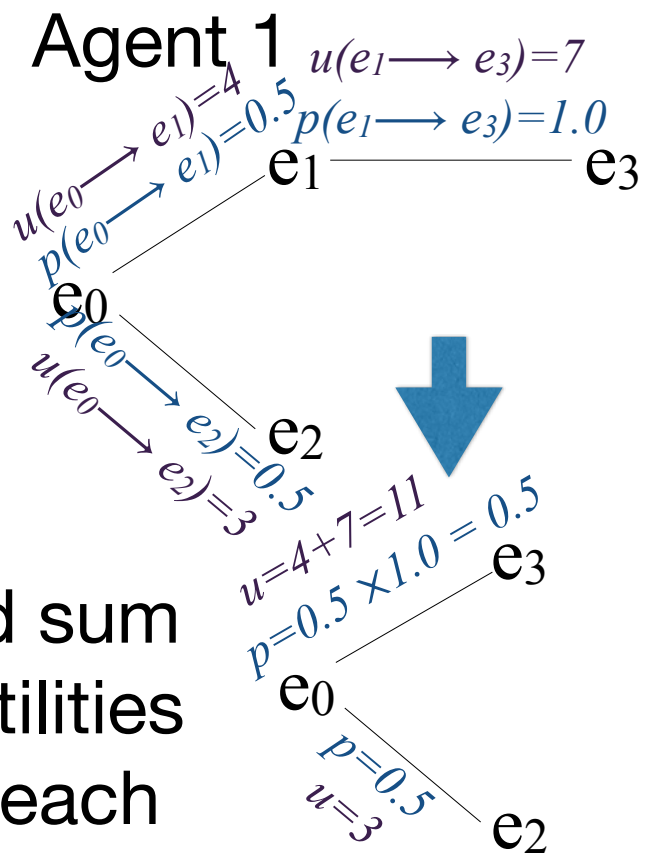
Example 2 solution



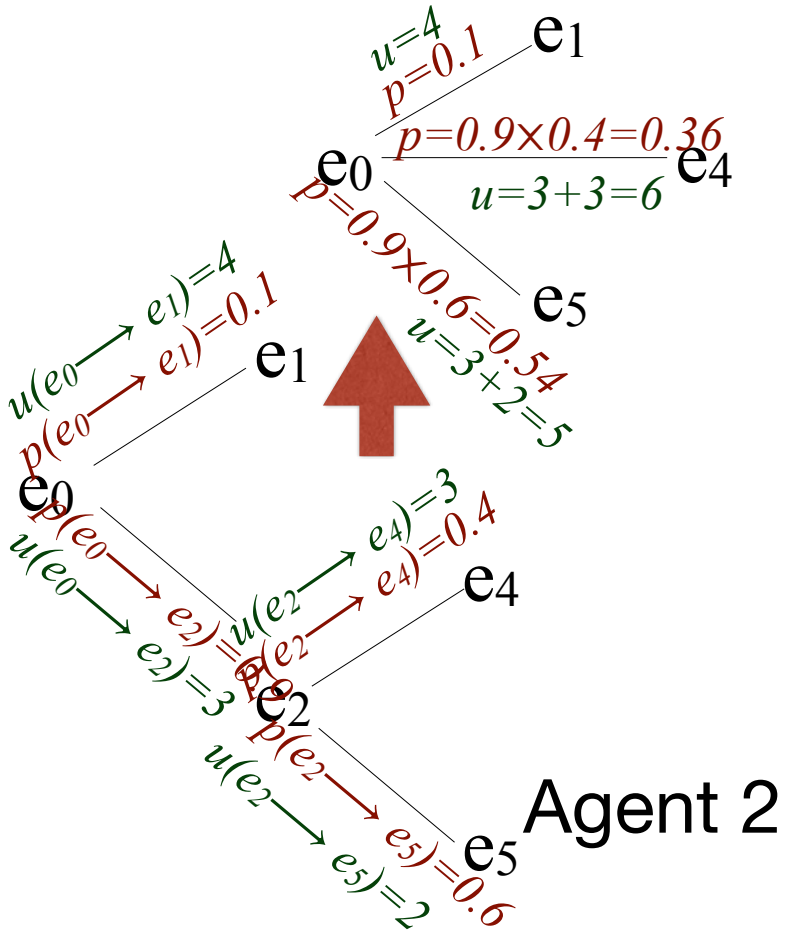
Example 2 solution



Example 2 solution



Find sum of utilities for each run



Example 2 solution

	Run	Utility	Probability
Agent 1	$e_0 \longrightarrow e_3$	$u=11$	$p=0.5$
	$e_0 \longrightarrow e_2$	$u=3$	$p=0.5$
Agent 2	$e_0 \longrightarrow e_1$	$u=4$	$p=0.1$
	$e_0 \longrightarrow e_4$	$u=6$	$p=0.36$
	$e_0 \longrightarrow e_5$	$u=5$	$p=0.54$

$$Ag_1 = (11 \times 0.5) + (3 \times 0.5) = 5.5 + 1.5 = 7$$

$$Ag_2 = (4 \times 0.1) + (6 \times 0.36) + (5 \times 0.54)$$
$$= 0.4 + 2.16 + 2.7 = 5.26$$

Bounded Optimal Agents

- Some agents cannot be implemented on some computers
  - The number of actions possible on an environment (and consequently the number of states) may be so big that it may need more than available memory to implement.

- We can therefore constrain our agent set to include only those agents that can be implemented on machine m:

$$\mathcal{AG}_m = \{Ag \mid Ag \in \mathcal{AG} \text{ and } Ag \text{ can be implemented on } m\}.$$

- The bounded optimal agent, Ag_{bopt}, with respect to m is then. . .

$$Ag_{bopt} = \arg \max_{Ag \in \mathcal{AG}_m} EU(Ag, Env)$$

Predicate Task Specifications

- A special case of assigning utilities to histories is to assign 0 (false) or 1 (true) to a run.
  - If a run is assigned 1, then the agent ***succeeds*** on that run, otherwise it ***fails***.

- Call these predicate task specifications.
  - Denote predicate task specification by Ψ:

$$\Psi : \mathcal{R} \rightarrow \{0, 1\}$$

Task Environments

- A task environment is a pair $<Env, \Psi>$, where Env is an environment, and the task specification Ψ is defined by:

$$\Psi : \mathcal{R} \rightarrow \{0, 1\}$$

- Let the set of all task environments be defined by: $\mathcal{TE}$

- A task environment specifies:
  - the properties of the system the agent will inhabit;
  - the criteria by which an agent will be judged to have either failed or succeeded.

Task Environments

- To denote set of all runs of the agent Ag in environment Env that satisfy Ψ, we write:

$$\mathcal{R}_{\Psi}(Ag, Env) = \{r \mid r \in \mathcal{R}(Ag, Env) \text{ and } \Psi(r) = 1\}.$$

- We then say that an agent Ag succeeds in task environment $<Env, \Psi>$ if

$$\mathcal{R}_{\Psi}(Ag, Env) = \mathcal{R}(Ag, Env)$$

- In other words, an agent succeeds if every run satisfies the specification of the agent.

We could also write this as:
$\forall r \in \mathcal{R}(Ag, Env), \text{we have} \Psi(r) = 1$

However, this is a bit ***pessimistic***: if

A more ***optimistic*** idea of success is:
$\exists r \in \mathcal{R}(Ag, Env), \text{we have} \Psi(r) = 1$

which counts an agent as successful as

The Probability of Success

- If the environment is non-deterministic, the τ returns a ***set*** of possible states.
  - We can define a probability distribution across the set of states.
  - Let $P(r \mid Ag, Env)$ denote probability that run r occurs if agent Ag is placed in environment Env.
  - Then the probability $P(\Psi \mid Ag, Env)$ that Ψ is satisfied by Ag in Env would then simply be:

$$P(\Psi \mid Ag, Env) = \sum_{r \in \mathcal{R}_\Psi(Ag, Env)} P(r \mid Ag, Env)$$

Achievement and Maintenance Tasks

- The idea of a predicate task specification is admittedly abstract.

- It generalises two common types of tasks, ***achievement tasks*** and ***maintenance tasks***:

    1. Achievement tasks: Are those of the form “achieve state of affairs φ”.
    2. Maintenance tasks: Are those of the form “maintain state of affairs ψ”.

Achievement and Maintenance Tasks

An ***achievement task*** is specified by a set G of "good" or "goal" states: $G \subseteq E$.

- The agent succeeds if it is guaranteed to bring about at least one of these states (we don't care which, as all are considered good).
- The agent ***succeeds*** if in an achievement task if it can ***force the environment*** into one of the goal states $g \in G$.

A ***maintenance goal*** is specified by a set B of "bad" states: $B \subseteq E$.

- The agent succeeds in a particular environment if it manages to avoid all states in B — if it never performs actions which result in any state in B occurring.
- In terms of games, the agent ***succeeds*** in a maintenance task if it ensures that it is ***never forced into*** one of the fail states $b \in B$.

Rationality

The agents rationality is given by the choice of actions based on expected utility of the outcome of the action. The rational agent selects an action a that provides the maximal expected outcome:

$$a = \arg\max_{l \in \mathcal{L}} \sum_{p_i : o_i \in l} p_i u(o_i)$$

Bounded Rationality: capability of the agent to perform rational decision (to choose the lottery providing maximal expected outcome) given bounds on computational resources:

- bounds on time complexity
- bounds on memory requirements

Calculative Rationality: capability to perform rational choice earlier than a fastest change in the environment can occur.

Rationality

Let us have a community of agents $A_j \in \mathcal{A}$ each choosing to play an action a_j, executing the lottery l_j. providing the agents with the utility $u(a_i)$.

- **Self-interested rational agent:** selects the action that optimizes its individual utility

$$a = \arg\max_{l \in \mathcal{L}} \sum_{p_i : o_i \in l} p_i U(o_i)$$

- **Cooperative rational agent:** selects the action that optimizes collective utility of the whole team:

$$a = \arg\max_{l \in \mathcal{L}} \sum_{\forall a_j \in \mathcal{A} - a} \sum_{p_{i,j} : o_{i,j} \in l_j} p_{i,j} u(o_{i,j}) + \sum_{p_i : o_i \in l} p_i u(o_i)$$