



# Logic for Multiagent Systems

Master 1st Year, 1st Semester 201/2022

Laurențiu Leuștean

Web page: <http://cs.unibuc.ro/~lleustean/>



# Multiagent systems



### Credits and acknowledgements

The lecture follows

- [1] Michael Wooldridge, [An Introduction to MultiAgent Systems](#), Second Edition, John Wiley & Sons, 2009
  - [2] Lecture slides/handouts, made available by Michael Wooldridge [here](#)
  - [3] Yoav Shoham, Kevin Leyton-Brown, [Multiagents Systems](#), Cambridge University Press, 2009
- 
- ▶ Most of the material on the slides is taken from [1,2]. Some material is taken from [3].
  - ▶ The figures are taken from [1,2].



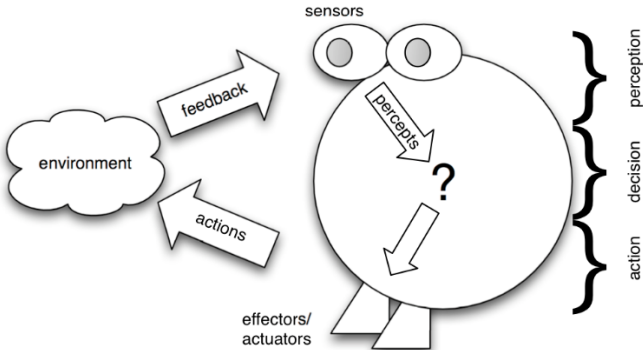
- ▶ The question **What is an agent?** does not have a definitive answer.
- ▶ Many competing, mutually inconsistent answers have been offered in the past.

### *Definition in [1,2]*

An **agent** is a computer system that is capable of **independent (autonomous) action** on behalf of its user or owner (figuring out what needs to be done to satisfy design objectives, rather than constantly being told).



## Agents



- ▶ An abstract view of an agent in its environment
- ▶ The agent takes sensory input from the environment, and produces, as output, actions that affect it. The interaction is usually an ongoing, non-terminating one.



### *Definition in [1,2]*

A **multiagent system** is one that consists of a number of agents, which **interact** with one-another.

Agents act on behalf of users with different goals and motivations. To successfully interact, they require the ability to **cooperate**, **coordinate**, and **negotiate** with each other, much as people do.

### *Definition in [3]*

A **multiagent system** is a system that includes multiple **autonomous** entities with either diverging information or diverging interests, or both.

The motivation for studying multiagent systems stems from interest in **artificial (software or hardware) agents**, for example software agents living on the Internet.



### *Examples*

- ▶ autonomous robots in a multi-robot setting
- ▶ trading agents
- ▶ game-playing agents that assist (or replace) human players in a multiplayer game
- ▶ interface agents - that facilitate the interaction between the user and various computational resources
- ▶ ...

The subject is highly **interdisciplinary**. Many of the ideas apply to inquiries about human individuals and institutions.



What is an intelligent agent? is another difficult question. There are some capabilities that we expect an intelligent agent to have.

### *Reactivity*

- ▶ Maintain an ongoing interaction with the environment,
- ▶ Respond to changes that occur in it (in time for the response to be useful).
- ▶ Most environments are **dynamic**.

### *Proactiveness*

- ▶ Goal directed behaviour.
- ▶ Generate and attempt to achieve goals.
- ▶ Take the initiative.
- ▶ Recognise opportunities.





### *Social ability*

- ▶ The ability to interact with other agents (and possibly humans) via **cooperation**, **coordination**, and **negotiation**.
- ▶ **Cooperation** is working together as a team to achieve a shared goal. It gives a better result.
- ▶ **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.
- ▶ **Negotiation** is the ability to reach agreements on matters of common interest. Typically involves offer and counter-offer, with compromises made by participants.



## *Abstract architectures for intelligent agents*

---

Make formal the abstract view of agents.

- ▶ Assume the **environment** may be in any of a finite set  $E$  of discrete, instantaneous **states**:

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

- ▶ An agent is assumed to have a repertoire of possible **actions** available:

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

- ▶ Actions transform the state of the environment.
- ▶ We assume that the set  $Ac$  of actions contains a **special action** *null*, with the meaning that nothing will be done.
- ▶ states are denoted also by  $e_0, e_1, \dots$
- ▶ actions are denoted also by  $\alpha_0, \alpha_1, \dots$ ,



## *Abstract architectures for intelligent agents*

---

The basic model of agents interacting with their environments is as follows:

- ▶ The environment starts in some state.
- ▶ The agent begins by choosing an action to perform on that state.
- ▶ As a result of this action, the environment can respond with a number of possible states. However, only one state will **actually** result — though, of course, the agent does not know in advance which it will be.
- ▶ On the basis of this second state, the agent again chooses an action to perform.
- ▶ The environment responds with one of a set of possible states.
- ▶ The agent then chooses another action; and so on.



## Abstract architectures for intelligent agents

A run over  $E$  and  $Ac$  is a finite sequence of interleaved environment states and actions.

### Definition 1.1

A *run*  $r$  over  $E$  and  $Ac$  is a finite sequence

$$r = (x_0, x_1, x_2, \dots, x_n),$$

where  $n \in \mathbb{N}$  and for all  $k \in \mathbb{N}$ ,  $x_{2k} \in E$  and  $x_{2k+1} \in Ac$ .

Runs are denoted by  $r, r', \dots$ . We write a run  $r$  as follows:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{u-2}} e_{u-1} \xrightarrow{\alpha_{u-1}} e_u$$

or

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{u-2}} e_{u-1} \xrightarrow{\alpha_{u-1}}$$



### *Notation 1.2*

- ▶  $\mathcal{R}$  denotes the set of all runs (over  $E$  and  $A_c$ ).
- ▶  $\mathcal{R}^{A_c}$  is the subset of these that end with an action.
- ▶  $\mathcal{R}^E$  is the subset of these that end with an environment state.



### Definition 1.3

A function  $\tau : \mathcal{R}^{Ac} \rightarrow 2^E$  is said to be a **state transformer** function.

- ▶ A state transformer function maps a run  $r \in \mathcal{R}^{Ac}$  to a set  $\tau(r)$  of possible environment states that could result from performing the action.
- ▶ State transformer functions represent the effect that an agent's actions have on an environment.

If  $\tau(r) = \emptyset$ , then there are no possible successor states to  $r$ . In this case, we say that the run  $r$  **has ended** or that  $r$  is a **terminated** run.

**Recall:** For any set  $A$ ,  $2^A$  is the set of all subsets of  $A$ :

$$2^A = \{B \mid B \subseteq A\}.$$



### Definition 1.4

An **environment** is a triple  $Env = (E, e_0, \tau)$ , where  $E$  is the set of environment states,  $e_0 \in E$  is an **initial state**, and  $\tau$  is a state transformer function.

Environments are:

- ▶ **history dependent**. The next state of an environment is not solely determined by the action performed by the agent and the current state of the environment. The actions made **earlier** by the agent also play a part in determining the current state.
- ▶ **non-deterministic**. There is **uncertainty** about the result of performing an action in some state.



We introduce a model of the agents that inhabit systems.

### Definition 1.5

An **agent** is a function  $Ag : \mathcal{R}^E \rightarrow Ac$  mapping runs (assumed to end with an environment state) to actions.

- ▶ An agent makes a decision about what action to perform based on the history of the system.
- ▶ Agents are deterministic.

### Definition 1.6

A **system** is a pair  $(Ag, Env)$  containing an agent  $Ag$  and an environment  $Env = (E, e_0, \tau)$ .





## Abstract architectures for intelligent agents

### Definition 1.7

A **run** in the system  $(Ag, Env)$  is a run  $r = (x_0, x_1, x_2, \dots, x_n)$  over  $E$  and  $Ac$  satisfying the following:

- ▶  $x_0 = e_0$ .
- ▶ for all  $k \geq 0$ :  
$$x_{2k+1} = Ag(x_0, \dots, x_{2k}) \text{ and } x_{2k+2} \in \tau(x_0, \dots, x_{2k+1}).$$
- ▶  $r$  is **terminated** in the following sense:
  - ▶ if  $x_n \in E$ , then  $\tau(r) = \emptyset$ ;
  - ▶ if  $x_n \in Ac$ , then  $Ag(r) = \text{null}$ .

We also say that  $r$  is a run of the agent  $Ag$  in the environment  $Env$ .

### Notation 1.8

We denote by  $\mathcal{R}(Ag, Env)$  the set of all runs in the system  $(Ag, Env)$ .



## *Abstract architectures for intelligent agents*

Let  $r \in \mathcal{R}^{Ac}$ ,

$$r = e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{u-2}} e_{u-1} \xrightarrow{\alpha_{u-1}}$$

Then  $r \in \mathcal{R}(Ag, Env)$  iff the following are satisfied:

- ▶  $\alpha_0 = Ag(e_0)$ .
- ▶ For all  $j = 1, \dots, u-1$ ,

$$e_j \in \tau(e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{j-2}} e_{j-1} \xrightarrow{\alpha_{j-1}})$$
$$\alpha_j = Ag(e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{j-1}} e_j).$$

- ▶  $\tau(r) = \emptyset$ .



## Abstract architectures for intelligent agents

Let  $r \in \mathcal{R}^E$ ,

$$r = e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{u-2}} e_{u-1} \xrightarrow{\alpha_{u-1}} e_u$$

Then  $r \in \mathcal{R}(Ag, Env)$  iff the following are satisfied:

- ▶  $\alpha_0 = Ag(e_0)$ .
- ▶ For all  $j = 1, \dots, u$ ,

$$e_j \in \tau(e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{j-2}} e_{j-1} \xrightarrow{\alpha_{j-1}})$$

$$\alpha_{j-1} = Ag(e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{j-2}} e_{j-1}).$$

- ▶  $Ag(r) = null$ .



### *Definition 1.9*

Two agents  $Ag_1$  and  $Ag_2$  are said to be

- ▶ *behaviourally equivalent with respect to environment  $Env$  if and only if  $\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$ .*
- ▶ *behaviourally equivalent* if they are behaviourally equivalent with respect to all environments.



## *Purely reactive agents*

---

- ▶ Certain types of 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**, since they simply respond directly to their environment.

### *Definition 1.10*

A **purely reactive agent** is a mapping  $Ag_{pure} : E \rightarrow Ac$ .



## Example: Thermostat

---

A **thermostat** is a very simple example of a **control system**.

- ▶ A thermostat has a sensor for detecting room temperature, and it produces as output one of two signals:
  - ▶ one that indicates that the **temperature is too low**;
  - ▶ another one which indicates that the **temperature is OK**.
- ▶ Its **delegated goal** is to maintain room temperature, the available **actions** being 'heating on' and 'heating off'.
- ▶ The **decision-making** component of the thermostat implements the following rules:

|                        |   |              |
|------------------------|---|--------------|
| temperature is too low | → | heating on,  |
| temperature is OK      | → | heating off. |



## Example: Thermostat

---

- ▶ Let us denote
$$e_1 := \text{temperature too low}, \quad e_2 := \text{temperature OK},$$
$$\alpha_1 := \text{heating on}, \quad \alpha_2 := \text{heating off}.$$
- ▶ Then  $E := \{e_1, e_2\}$  and  $Ac := \{\alpha_1, \alpha_2\}$ .
- ▶ The thermostat is the purely reactive agent *Therm* defined as follows:

$$Therm : E \rightarrow Ac, \quad Therm(e) = \begin{cases} \alpha_1 & \text{if } e = e_1, \\ \alpha_2 & \text{if } e = e_2. \end{cases}$$



## *Intelligent agents*

---

This view of agents is too abstract. It does not help us to **construct** them, since it gives us no clues about how to design the decision function **action**.

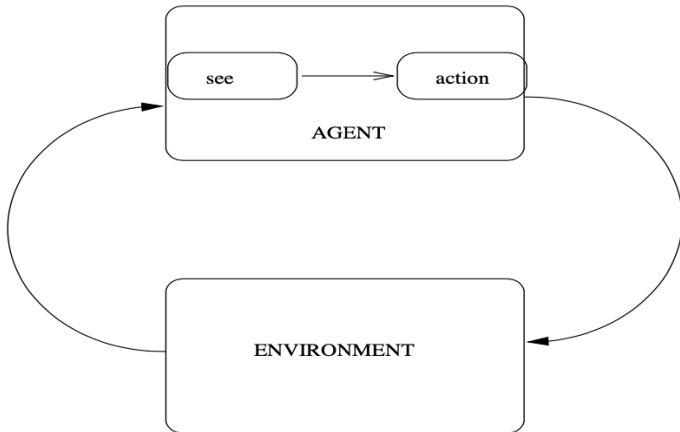
- ▶ We **refine** our abstract model of agents, by breaking it down into **subsystems**.
- ▶ We make **design choices** on these subsystems — what data and control structures will be present.
- ▶ An **agent architecture** is essentially a map of the internals of an agent — its data structures, the operations that may be performed on these data structures, and the control flow between these data structures.
- ▶ There are different types of agent architectures, with very different views on the data structures and algorithms that will be present within an agent.





## Perception

One high-level design decision is the separation of an agent's decision function into **perception** and **action** subsystems.





## Perception

- ▶ The **perception** function **see** captures the agent's ability to observe its environment. **Example**: a video camera or an infra-red sensor on a mobile robot.
- ▶ The output of the **see** function is a **percept** — a perceptual input.
- ▶ The **action** function represents the agent's decision making process.

Let **Per** be a nonempty set of **percepts**.

### Definition 1.11

The **see** and **action** functions are defined as follows:

$$\text{see} : E \rightarrow \text{Per} \quad \text{and} \quad \text{action} : \text{Per}^* \rightarrow \text{Ac}.$$

**Recall**: For any set  $A$ ,  $A^*$  is the set of all finite sequences of elements of  $A$ :

$$A^* = \{a_1 a_2 \dots a_n \mid n \in \mathbb{N} \text{ and } a_i \in A \text{ for all } i = 1, \dots, n\}.$$



These simple definitions allow us to explore some interesting properties of agents and perception.

Suppose that we have two environment states  $e_1, e_2 \in E$  such that  $e_1 \neq e_2$ , but  $\text{see}(e_1) = \text{see}(e_2)$ . Then  $e_1$  and  $e_2$  are mapped to the same percept, and the agent receives the same perceptual information from each of them. As far as the agent is concerned,  $e_1$  and  $e_2$  are **indistinguishable**.

### Definition 1.12

The relation  $\equiv$  on  $E$  is defined as follows: for every  $e_1, e_2 \in E$ ,

$$e_1 \equiv e_2 \quad \text{iff} \quad \text{see}(e_1) = \text{see}(e_2).$$

### Remark 1.13

$\equiv$  is an equivalence relation on  $E$ .



- ▶  $\equiv$  partitions  $E$  into mutually indistinguishable sets of states, namely the different equivalence classes  $[e]$ , where  $e \in E$ .
- ▶ If  $[e] = \{e\}$  for every  $e \in E$ , then  $see(e_1) \neq see(e_2)$  for every states  $e_1 \neq e_2$ . The agent **can** distinguish **every** state — the agent has perfect perception in the environment.
- ▶ If  $[e] = E$  for every  $e \in E$ , then  $see(e_1) = see(e_2)$  for every states  $e_1, e_2$ . The agent's perceptual ability is non-existent, it **cannot** distinguish between **any** different states. As far as the agent is concerned, all environment states are identical.



## Example

---

- ▶ Let us consider the statements

$x :=$  “the room temperature is OK”

$y :=$  “Angela Merkel is the German Chancellor”

- ▶ Assume that these are the only two facts about our environment that we are concerned with. Then

$$E = \{e_1 := \{x, y\}, e_2 := \{x, \neg y\}, e_3 := \{\neg x, y\}, e_4 := \{\neg x, \neg y\}\}$$

- ▶ In state  $e_1$  the room temperature is OK and Angela Merkel is the German Chancellor; in state  $e_2$  the room temperature is OK and Angela Merkel is not the German Chancellor; and so on.
- ▶ The thermostat is sensitive **only** to temperatures in the room.
- ▶ The room temperature is not casually related to whether or not Angela Merkel is the German Chancellor.
- ▶ The states where Angela Merkel is and is not the German Chancellor are **indistinguishable** to the thermostat.



## Perception - an example

---

The set of percepts is  $Per = \{p_1, p_2\}$ , where

$p_1 := \text{temperature OK}$ ,       $p_2 := \text{temperature too low}$ .

The perception function  $see$  is defined as follows:

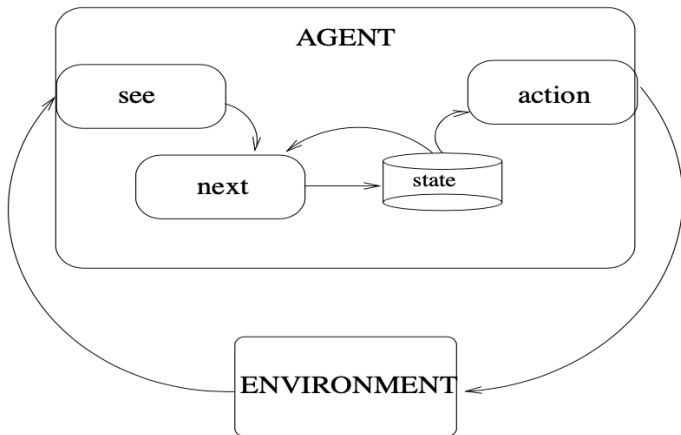
$$see : E \rightarrow Per, \quad see(e) = \begin{cases} p_1 & \text{if } e = e_1 \text{ or } e = e_2, \\ p_2 & \text{if } e = e_3 \text{ or } e = e_4. \end{cases}$$

- ▶  $[e_1] = [e_2] = \{e_1, e_2\}$
- ▶  $[e_3] = [e_4] = \{e_3, e_4\}$



## *Agents with state*

We now consider agents that **maintain state**.





## Agents with state

---

- ▶ These agents have 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.

### Definition 1.14

The **see** and **action** functions are defined as follows:

$$\text{see} : E \rightarrow \text{Per} \quad \text{and} \quad \text{action} : I \rightarrow \text{Ac}.$$

The perception function *see* is unchanged. The action-selection function *action* takes as inputs internal states.

### Definition 1.15

The function **next** is defined as follows:

$$\text{next} : I \times \text{Per} \rightarrow I.$$





### *The behaviour of a state-based agent:*

- ▶ The agent starts in some initial internal state  $i_0$ .
- ▶ It then observes its environment state  $e$ , and generates a percept  $see(e)$ .
- ▶ The internal state of the agent is then updated to  $i_1 := next(i_0, see(e))$ .
- ▶ The action selected by the agent is  $\alpha := action(i_1)$ .
- ▶ The agent performs action  $\alpha$ .
- ▶ The agent enters another cycle: perceives the world via  $see$ , updates its state via  $next$ , and chooses an action to perform via  $action$ .



## *Tasks for agents*

---

- ▶ We build agents in order to carry out **tasks** for us.
- ▶ The tasks to be carried out must be **specified** by us in some way
- ▶ How to specify these tasks? How to tell the agent what to do?

One way to to do this: write a program for the agent to execute.

- ▶ Advantage: no uncertainty about what the agent will do. It will do exactly what we told it to, and no more.
- ▶ Disadvantage: if unforeseen circumstances arise, the agent executing the task will be unable to respond accordingly.



## *Tasks for agents*

---

- ▶ We want to tell our agent what to do without telling it how to do it.
- ▶ One way of doing this is to define tasks indirectly, via some kind of performance measure.
- ▶ One possibility: associate utilities with states of the environment.
- ▶ A utility is a numeric value representing how 'good' a state is: the higher the utility, the better.
- ▶ The task of the agent is then to bring about states that maximize utility.
- ▶ We do not specify to the agent how this is to be done.



### Definition 1.16

A *utility function* (or *task specification*) is a function  $u : E \rightarrow \mathbb{R}$ .

What is the *overall utility* of a *run*?

- ▶ minimum utility of a state on run?
- ▶ maximum utility of a state on run?
- ▶ sum of utilities of all states on run?
- ▶ average utility of all states on run?

Main disadvantage:

- ▶ assigns utilities to *local states*.
- ▶ difficult to specify a *long-term* view when assigning utilities to individual states.



Idea: assign a utility not to individual states, but to **runs**.

### Definition 1.17

A **utility function** (or **task specification**) is a function  $u : \mathcal{R} \rightarrow \mathbb{R}$ .

- ▶ If we are concerned with agents that must operate independently over long periods of time, then this approach is appropriate.
- ▶ The utility-based approach works well in certain scenarios.
- ▶ Problems:
  - ▶ Sometimes it is difficult to define a utility function.
  - ▶ People don't think in terms of utilities. It is hard for people to specify tasks in these terms.



Tileworld was proposed as an experimental environment for evaluating agent architectures.

- ▶ 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**, and if it is located **next** to a tile, it can **push** it.
- ▶ An **obstacle** is a group of immovable grid cells.
- ▶ **Holes** have to be filled up with tiles by the agent.
- ▶ An agent scores **points** by **filling** holes with tiles, the aim being to fill **as many holes as possible**.
- ▶ Holes appear **randomly** and exist for as long as their **life expectancy**, unless they disappear because of the agent's actions.
- ▶ The interval between the appearance of successive holes is called the hole **gestation time**.



- ▶ Tileworld is an example of a **dynamic** environment: starting in some **randomly** generated world state, based on **parameters** set by the **experimenter**, it changes over time in discrete steps, with the random appearance and disappearance of holes.
- ▶ The **performance** of an agent in the Tileworld is measured by **running** the Tileworld testbed for a predetermined number of time steps, and **measuring** the number of holes that the agent succeeds in filling.
- ▶ Experimental **error** is eliminated by running the agent in the environment a number of times, and computing the **average** of the performance.



## Definition 1.18

The *utility function* is defined as follows:

$$u : \mathcal{R} \rightarrow \mathbb{R}, \quad u(r) = \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

- ▶  $u$  is normalized:  $u(r) \in [0, 1]$  for every run  $r$
- ▶  $u(r) = 1$ : agent filled every hole that appeared in  $r$
- ▶  $u(r) = 0$ : agent did not fill any hole that appeared in  $r$





- ▶ Despite its simplicity, Tileworld allows us to examine several important capabilities of agents.
- ▶ Examples of abilities of agents:
  - ▶ to **react** to changes in the environment
  - ▶ to **exploit opportunities** when they arise.

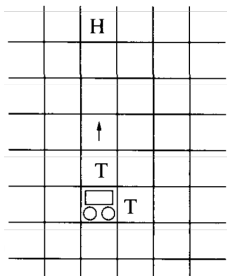


Figure 1: Tileworld example

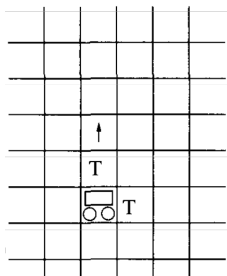


Figure 2: Tileworld example

## Example 1.19

Suppose an agent is pushing a tile to a hole (Figure 1), when this hole disappears (Figure 2).

The agent should recognize this change in the environment, and modify its behaviour appropriately.

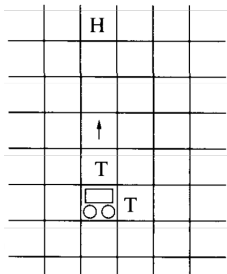


Figure 3: Tileworld example

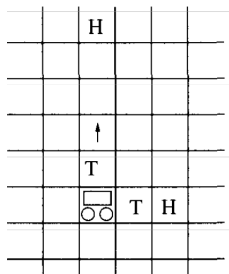


Figure 4: Tileworld example

### Example 1.20

Suppose an agent is pushing a tile to a hole (Figure 3), when a hole appears to the right of the agent (Figure 4).

It would do better to push the tile to the right, than to continue to head north, for the simple reason that it only has to push the tile one step, rather than three.

### Notation 1.22

$P(r \mid Ag, Env)$  denotes the probability that run  $r$  occurs when agent  $Ag$  is placed in environment  $Env$ .

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

### Definition 1.23

The **expected utility** of agent  $Ag$  in environment  $Env$  (given  $P, u$ ) is defined as follows:

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



## Expected utility - an example

Consider the environment  $(E, e_0, \tau)$  defined as follows:

►  $E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$  and  $Ac = \{\alpha_0, \alpha_1, null\}$ .

► Define

$$\tau : \mathcal{R}^{Ac} \rightarrow 2^E, \quad \tau(r) = \begin{cases} \{e_1, e_2\} & \text{if } r = e_0 \xrightarrow{\alpha_0}, \\ \{e_3, e_4, e_5\} & \text{if } r = e_0 \xrightarrow{\alpha_1}, \\ \emptyset & \text{otherwise.} \end{cases}$$

We consider the following two agents in this environment:

$$Ag_1 : \mathcal{R}^E \rightarrow Ac, \quad Ag_1(r) = \begin{cases} \alpha_0 & \text{if } r = e_0 \\ null & \text{otherwise} \end{cases}$$

$$Ag_2 : \mathcal{R}^E \rightarrow Ac, \quad Ag_2(r) = \begin{cases} \alpha_1 & \text{if } r = e_0 \\ null & \text{otherwise} \end{cases}$$



## Expected utility - an example

*Runs of  $Ag_1$  and  $Ag_2$  in the environment  $Env$*

$$\mathcal{R}(Ag_1, Env) = \{r_1 := e_0 \xrightarrow{\alpha_0} e_1, r_2 := e_0 \xrightarrow{\alpha_0} e_2\},$$

$$\mathcal{R}(Ag_2, Env) = \{r_3 := e_0 \xrightarrow{\alpha_1} e_3, r_4 := e_0 \xrightarrow{\alpha_1} e_4, r_5 := e_0 \xrightarrow{\alpha_1} e_5\}.$$

The probabilities of the various runs are defined as follows:

► for the agent  $Ag_1$ :

$$P(r_1 \mid Ag_1, Env) = 0.4, \quad P(r_2 \mid Ag_1, Env) = 0.6.$$

► for the agent  $Ag_2$ :

$$P(r_3 \mid Ag_2, Env) = 0.1, \quad P(r_4 \mid Ag_2, Env) = 0.2, \\ P(r_5 \mid Ag_2, Env) = 0.7.$$



## *Expected utility - an example*

---

Assume the utility function  $u$  is defined as follows:

$$\begin{aligned}u(r_1) &= 8, & u(r_2) &= 11, \\u(r_3) &= 70, & u(r_4) &= 9, & u(r_5) &= 10.\end{aligned}$$

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

$$\begin{aligned}EU(Ag_1, Env) &= u(r_1)P(r_1 \mid Ag_1, Env) + u(r_2)P(r_2 \mid Ag_1, Env) \\&= 8 \times 0.4 + 11 \times 0.6 = 9.6\end{aligned}$$

$$\begin{aligned}EU(Ag_2, Env) &= u(r_3)P(r_3 \mid Ag_2, Env) + u(r_4)P(r_4 \mid Ag_2, Env) \\&\quad + u(r_5)P(r_5 \mid Ag_2, Env) \\&= 70 \times 0.1 + 9 \times 0.2 + 10 \times 0.7 = 15.7\end{aligned}$$

### Notation 1.24

Let  $AG$  denote the set of all agents acting in some environment.

Assume that the utility function  $u$  is **bounded from above**, that is: there exists  $M \in \mathbb{R}$  such that  $u(r) \leq M$  for all  $r \in \mathcal{R}$ .

### Definition 1.25

An **optimal agent** in an environment  $Env$  is an agent  $Ag_{opt}$  that maximizes the expected utility:

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

- ▶ 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.
- ▶ The definition does not not give us any clues about how to **implement** this agent.
- ▶ There are agents that **cannot** be implemented on a real computer.





## Bounded optimal agents

---

Suppose  $m$  is a particular computer.

### Notation 1.26

$AG_m$  denotes the set of agents that can be implemented on  $m$ :

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

### Definition 1.27

A **bounded optimal agent** in an environment  $Env$ , with respect to  $m$ , is an agent  $Ag_{bopt}$  that maximizes the expected utility:

$$Ag_{bopt} = \arg \max_{Ag \in AG_m} EU(Ag, Env).$$

- We consider only the agents that can actually be implemented on the machine that we have for the task.



## Predicate task specifications

---

- ▶ A **predicate task specification** is one where the utility function acts as a **predicate** over runs.
- ▶ A utility function  $u : \mathcal{R} \rightarrow \mathbb{R}$  is a **predicate** if the range of  $u$  is the set  $\{0, 1\}$ , that is if  $u$  assigns a run either 1 (true) or 0 (false).
- ▶ If  $u(r) = 1$ , we say that the run  $r$  **satisfies** the specification; the agent **succeeds** on the run  $r$ .
- ▶ If  $u(r) = 0$ , we say that the run  $r$  **fails to satisfy** the specification; the agent **fails** on the run  $r$ .

### Definition 1.28

A **predicate task specification** is a mapping  $\Psi : \mathcal{R} \rightarrow \{0, 1\}$ .



### Definition 1.29

A **task environment** is a pair  $(Env, \Psi)$ , where  $Env$  is an environment, and  $\Psi$  is a predicate task specification.

### Notation 1.30

$TE$  denotes the set of all task environments.

A task environment specifies:

- ▶ the properties of the system the agent will inhabit (i.e. the environment  $Env$ );
- ▶ the criteria by which an agent will be judged to have either failed or succeeded (i.e. the specification  $\Psi$ ).



### Notation 1.31

$\mathcal{R}_\Psi(\text{Ag}, \text{Env})$  denotes the set of all runs of agent  $\text{Ag}$  that satisfy  $\Psi$ :

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

There are more possibilities to define the success of an agent in a task environment.

### *The pessimistic definition:*

We say that an agent  $\text{Ag}$  **succeeds** in task environment  $(\text{Env}, \Psi)$  if  $\mathcal{R}_\Psi(\text{Ag}, \text{Env}) = \mathcal{R}(\text{Ag}, \text{Env})$ .

Thus, the agent succeeds iff every possible run of the agent in the environment satisfies the predicate task specification.



## Task environment

---

### *The optimistic definition:*

We say that an agent  $Ag$  **succeeds** in task environment  $(Env, \Psi)$  if  $\mathcal{R}_{\Psi}(Ag, Env) \neq \emptyset$ .

Thus, the agent succeeds iff **at least one** run of the agent in the environment satisfies the predicate task specification.

### *The probabilistic definition:*

The **success** of an agent  $Ag$  in task environment  $(Env, \Psi)$  is defined as the probability  $P(\Psi|Ag, Env)$  that the predicate task specification  $\Psi$  is satisfied by the agent in the environment  $Env$ .

### *Remark 1.32*

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



## *Achievement and maintenance tasks*

---

- ▶ The notion of a predicate task specification may seem rather abstract.
- ▶ It is a generalization of certain very common forms of tasks.

Two most common types of tasks are **achievement tasks** and **maintenance tasks**:

- ▶ Achievement tasks are those of the form 'achieve state of affairs  $\varphi$ '.
- ▶ Maintenance tasks are those of the form 'maintain state of affairs  $\varphi$ '.



### Definition 1.33

The task environment  $(Env, \Psi)$  specifies an **achievement task** if there exists some set of states  $G \subseteq E$  such that for all  $r \in \mathcal{R}(Ag, Env)$ ,

$\Psi(r) = 1$  iff there exists some state  $e \in G$  such that  $e$  occurs in  $r$ .

We also say that  $(Env, \Psi)$  is an **achievement task environment**.

The elements of  $G$  are the **goal states** of the task.

### Notation 1.34

We use  $(Env, G)$  to denote an achievement task environment with goal states  $G$  and environment  $Env$ .

An agent is successful if is guaranteed to bring about one of the goal states (we do not care which one — all are considered equally good).



## *Achievement tasks*

---

A useful way to think about achievement tasks is as the agent **playing a game** against the environment:

- ▶ The environment and agent both begin in some state.
- ▶ The agent executes an action, and the environment responds with some state.
- ▶ The agent then executes another action, and so on.
- ▶ The agent **'wins'** if it can force the environment into one of the goal states.





## Maintenance tasks

---

- ▶ Many other tasks can be classified as problems where the agent is required to **avoid** some state of affairs.
- ▶ We refer to such tasks as **maintenance** tasks.

### Definition 1.35

The task environment  $(Env, \Psi)$  specifies a **maintenance task** if there exists some set of states  $B \subseteq E$  such that for all  $r \in \mathcal{R}(Ag, Env)$ ,

$$\Psi(r) = 1 \text{ iff for all states } e \in B, e \text{ does not occur in } r.$$

We also say that  $(Env, \Psi)$  is a **maintenance task environment**.

The elements of  $B$  are the **bad states** of the task.

### Notation 1.36

We use  $(Env, B)$  to denote a maintenance task environment with bad states  $B$  and environment  $Env$ .



It is again useful to think of maintenance tasks as games:

- ▶ The agent '**wins**' if it manages to avoid all the bad states.
- ▶ The environment, in the role of opponent, is attempting to force the agent into  $B$ .
- ▶ The agent is successful if it has a winning strategy for avoiding  $B$ .



## *Achievement and maintenance tasks*

---

- ▶ More complex tasks might be specified by **combinations** of achievement and maintenance tasks.
- ▶ A simple combination:  
achieve any one of states  $G$  while avoiding all states  $B$ .



- ▶ Knowing that there exists an agent which will succeed in a given task environment is helpful.
- ▶ However, it would be more helpful if, knowing this, we obtain such an agent, we implement it.
- ▶ How do we do this?
- ▶ An obvious answer is: '**manually**' **implement** the agent from the specification.

There is at least one other possibility:

develop an algorithm that will **automatically synthesize** such agents for us from task environment specifications.



**Agent synthesis** is automatic programming: the goal is to have a **program** that will take as input a task environment, and from this task environment automatically generate an agent that succeeds in this environment.

### Definition 1.37

An **agent synthesis algorithm** is a function

$$\text{syn} : TE \rightarrow AG \cup \{\perp\}.$$

A synthesis algorithm is

- ▶ **sound** if, whenever it returns an agent, this agent succeeds in the task environment that is passed as input, and
- ▶ **complete** if it is guaranteed to return an agent whenever there exists an agent that will succeed in the task environment given as input.



### Definition 1.38

An agent synthesis algorithm  $\text{syn}$  is

- ▶ **sound** if for any task environment  $(\text{Env}, \Psi) \in \text{TE}$ ,  
 $\text{syn}((\text{Env}, \Psi)) = \text{Ag}$  **implies**  $\mathcal{R}(\text{Ag}, \text{Env}) = \mathcal{R}_{\Psi}(\text{Ag}, \text{Env})$ .
- ▶ **complete** if for any  $(\text{Env}, \Psi) \in \text{TE}$ ,  
(there exists an agent  $\text{Ag}$  s.t.  $\mathcal{R}(\text{Ag}, \text{Env}) = \mathcal{R}_{\Psi}(\text{Ag}, \text{Env})$ )  
**implies**  $\text{syn}((\text{Env}, \Psi)) \neq \perp$ .
- ▶ Soundness ensures that a synthesis algorithm always delivers agents that do their job correctly, but may not always deliver agents, even where such agents are in principle possible.
- ▶ Completeness ensures that an agent will always be delivered where such an agent is possible, but does not guarantee that these agents will do their job correctly.



- ▶ Soundness and completeness ensure that a synthesis algorithm will output  $\perp$  iff there is no agent that does the job correctly.
- ▶ Ideally, we seek synthesis algorithms that are both sound and complete.
- ▶ Of the two conditions, soundness is probably the more important.
- ▶ There is not much point in complete synthesis algorithms that deliver 'buggy' agents.



# Agent architectures





- ▶ An **agent architecture** is a **software design** for an agent.
- ▶ We have already seen a top-level decomposition into:  
perception – state – decision – action
- ▶ An agent architecture defines:
  - ▶ key data structures;
  - ▶ operations on data structures;
  - ▶ control flow between operations.



Some types of agents:

- ▶ **logic-based** or **deductive reasoning** or **symbolic reasoning** agents - decision making is realised through logical deduction;
- ▶ **practical reasoning** agents - reasoning directed towards actions, the process of figuring out what to do; action selection through deliberation and means-end reasoning;
- ▶ **reactive** agents - reacting to an environment, without reasoning about it; no representation, direct link from perceptual input to action;
- ▶ **hybrid** agents - combine reactive and deliberative reasoning, usually in layered architecture.



## *Logic-based agents*

---

The **logic-based** approach is the classical approach to building agents.

### *Key ideas:*

- ▶ give a **symbolic representation** of the environment - **logical formulas**.
- ▶ manipulate **syntactically** this representation - **logical deduction** or **theorem proving**.

### *Problems to be solved:*

- ▶ **Transduction problem**: the problem of translating the real world into an accurate, adequate symbolic description of the world, in time for that description to be useful.
- ▶ **Representation/reasoning problem**: the problem of representing information symbolically, and getting agents to manipulate/reason with it, in time for the results to be useful.



## Agents as theorem provers

---

**Deliberate agents** are a simple model of logic-based agents.

- ▶ An **internal state** of such an agent is a **database** of formulas of **classical first-order logic**.
- ▶ The agent's database might contain formulas such as *Open(valve1)*, *Temperature(reactor6, 32)*, *Pressure(tank6, 28)*.
- ▶ The database is the **information** that the agent has about its environment.
- ▶ An agent's database plays a somewhat analogous role to that of **belief** in humans.
- ▶ Some facts from the database could be wrong - agent's sensors may be faulty, its reasoning may be faulty, the information may be out of date.
- ▶ Thus, the fact that *Open(valve1)* is in the database does not mean that *valve1* is open; it could be closed.



## Agents as theorem provers

---

We use the model of **agents with state**.

Let  $\mathcal{L}$  be a first-order language and  $Form_{\mathcal{L}}$  be the set of its formulas.

We assume that  $\mathcal{L}$  contains:

- ▶ a unary relation symbol  $Do$ ;
- ▶ a constant symbol  $c_{\alpha}$  for every action  $\alpha \in Ac$ . For simplicity, we write  $\alpha$  instead of  $c_{\alpha}$ .
- ▶ A **database** is a set of formulas of  $\mathcal{L}$ .
- ▶ Let  $\mathcal{D}$  be the set of all databases. Thus,  $\mathcal{D} = 2^{Form_{\mathcal{L}}}$ .
- ▶ We write  $DB, DB_1, \dots$  for members of  $\mathcal{D}$ .
- ▶ An internal state of the agent is a database. Thus,  $I = \mathcal{D}$ .



## Agents as theorem provers

---

- ▶ We fix a set  $\Sigma \subseteq \text{Form}_{\mathcal{L}}$  of formulas of  $\mathcal{L}$ , whose elements are called **deduction formulas**.
- ▶ We use the notation  $DB \vdash_{\Sigma} \varphi$  for  $DB \cup \Sigma \vdash \varphi$ .
- ▶ The idea is that  
if  $DB \vdash_{\Sigma} Do(\alpha)$ , then  $\alpha$  is the action to be performed by the agent.
- ▶ The agent's behaviour is determined by its deduction formulas (its '**program**') and its current database.

An agent's **action** selection function

$$action : \mathcal{D} \rightarrow Ac$$

is defined in terms of its deduction formulas. The pseudo-code definition of this function is given in Figure 5.



## Agents as theorem provers

---

1. function *action*( $DB : \mathcal{D}$ ) returns an action  $Ac$
2. begin
3.     for each  $\alpha \in Ac$  do
4.         if  $DB \vdash_{\Sigma} Do(\alpha)$  then
5.             return  $\alpha$
6.         end-if
7.     end-for
8.     for each  $\alpha \in Ac$  do
9.         if  $DB \not\vdash_{\Sigma} \neg Do(\alpha)$  then
10.             return  $\alpha$
11.         end-if
12.     end-for
13.     return *null*
14. end function *action*

Figure 5: Agent selection as theorem proving.



## *Agents as theorem provers*

---

- ▶ In lines 3-7, the agent takes each of its possible actions  $\alpha$  in turn, and attempts to prove the formula  $Do(\alpha)$  from its database  $DB$  (passed as a parameter to the function) using its set  $\Sigma$  of deduction formulas. If the agent succeeds in proving  $Do(\alpha)$ , then  $\alpha$  is returned as the action to be performed.
- ▶ If the agent fails to prove  $Do(\alpha)$ , for all actions  $\alpha \in Ac$ , then it tries to find an action that is consistent with the deduction formulas and the database, that is not explicitly forbidden.
- ▶ In lines 8-12, the agent attempts to find an action  $\alpha \in Ac$  such that  $\neg Do(\alpha)$  cannot be derived from its database using its deduction formulas. If it can find such an action, then this is returned as the action to be performed.
- ▶ If, however, the agent fails to find an action that is at least consistent, then it returns the special action *null*, indicating that no action has been selected.





The perception function *see* remains unchanged:

$$see : E \rightarrow Per,$$

where *Per* is the set of percepts.

The *next* function has the form:

$$next : \mathcal{D} \times Per \rightarrow \mathcal{D}.$$

It maps a database and a percept to a new database.



## *Agents as theorem provers - example*

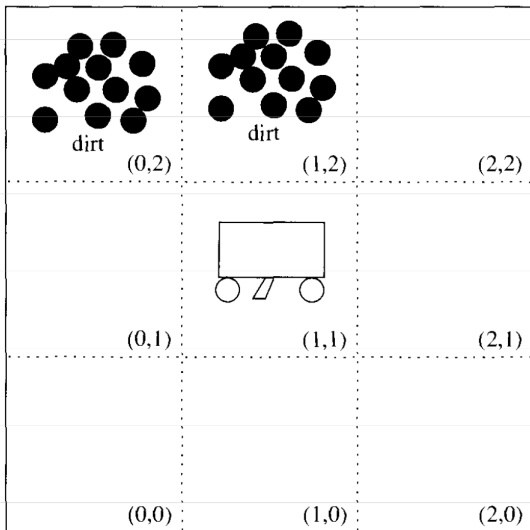
---

We consider an example: **vacuum cleaning world**

- ▶ We have a small **robotic agent** that will clean up a house.
- ▶ The robot is equipped with a **sensor** that will tell it whether it is over any **dirt**, and a **vacuum cleaner** that can be used to **suck up dirt**.
- ▶ The robot always has a definite orientation (one of **north**, **south**, **east**, or **west**).
- ▶ In addition to being able to suck up dirt, the agent can move **forward one 'step'** or **turn right 90 degrees**.
- ▶ The agent moves around a room, which is divided grid-like into a number of equally sized squares.
- ▶ Our agent does nothing but clean — it never leaves the room.
- ▶ Assume, for simplicity, that the room is a  **$3 \times 3$  grid**, and the agent always **starts in grid square (0, 0) facing north**.



## *Agents as theorem provers - example*





## Agents as theorem provers - example

---

- ▶ The set  $Per$  of **percepts** is defined as

$$Per = \{dirt, nothing\},$$

where **dirt** signifies that there is dirt beneath it, and **nothing** indicates no special information.

- ▶ The set  $Ac$  of actions is defined as

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

where **forward** means 'go forward', **suck** means 'suck up dirt', and **turn** means 'turn right 90 degrees'.

- ▶ The goal is to traverse the room continually searching for and removing dirt.



## Agents as theorem provers - example

---

We use three simple **domain predicates**:

$In(i, j)$       agent is at  $(i, j)$ ,

$Dirt(i, j)$     there is dirt at  $(i, j)$ ,

$Facing(d)$     the agent is facing direction  $d$ ,

where  $i, j \in \{0, 1, 2\}$  and  $d \in \{north, south, east, west\}$ .

This means that the first-order language  $\mathcal{L}$  contains:

- ▶ two binary relation symbols  $In$  and  $Dirt$ ;
- ▶ a unary relation symbol  $Facing$ ;
- ▶ constant symbols  $north, south, east, west$ ;
- ▶ constant symbols  $(i, j)$  for every  $i, j \in \{0, 1, 2\}$ .



## Agents as theorem provers - example

- ▶ The *next* function looks at the perceptual information obtained from the environment and at the actual database, and generates a new database which includes this information.
- ▶ It removes old or irrelevant information, and also, it tries to figure out the *new* location and orientation of the agent.
- ▶ We specify the *next* function in several parts.

Let *old*(DB) denote the set of 'old' information in a database, which we want the update function *next* to remove.

$$\textit{old}(DB) = DB \cap \Delta,$$

where

$$\begin{aligned} \Delta = & \{ \textit{In}(i, j) \mid i, j \in \{0, 1, 2\} \} \cup \{ \textit{Dirt}(i, j) \mid i, j \in \{0, 1, 2\} \} \\ & \cup \{ \textit{Facing}(d) \mid d \in \{ \textit{north}, \textit{south}, \textit{east}, \textit{west} \} \}. \end{aligned}$$



## Agents as theorem provers - example

- ▶ We require a function **new**, which gives the set of new formulas to add to the database:

$$\text{new} : \mathcal{D} \times \text{Per} \rightarrow \mathcal{D}.$$

- ▶ It must generate formulas
  - ▶ *In(...)*, describing the new position of the agent;
  - ▶ *Facing(...)* describing the orientation of the agent;
  - ▶ *Dirt(...)* if dirt has been detected at the new position.

The **next** function is defined as follows:

$$\text{next} : \mathcal{D} \times \text{Per} \rightarrow \mathcal{D}, \quad \text{next}(\text{DB}, p) = (\text{DB} - \text{old}(\text{DB})) \cup \text{new}(\text{DB}, p)$$



## Agents as theorem provers - example

---

The **deduction formulas** have the general form

$$\varphi \rightarrow \psi, \quad \text{where } \varphi, \psi \text{ are formulas of } \mathcal{L}$$

### *Cleaning*

$$In(x, y) \wedge Dirt(x, y) \rightarrow Do(suck) \quad x, y \text{ are variables}$$

- ▶ If the agent is at location  $(x, y)$  and it perceives dirt, then the prescribed action will be to suck up dirt.
- ▶ It takes priority over all other possible behaviours of the agent (such as navigation).





## Agents as theorem provers - example

### Traversal

- ▶ The basic action of the agent is to traverse the world.
- ▶ For simplicity, we assume that the robot will always move from (0,0) to (0,1) to (0,2) and then to (1,2), (1,1) and so on. Once the agent reaches (2,2), it must head back to (0,0).
- ▶ The deduction formulas dealing with the traversal up to (0,2):

$$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \rightarrow Do(forward)$$

$$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \rightarrow Do(forward)$$

$$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \rightarrow Do(turn)$$

$$In(0,2) \wedge Facing(east) \rightarrow Do(forward)$$

- ▶ Similar formulas can be easily generated that will get the agent to (2,2) and back to (0,0).



## Logic-based agents

---

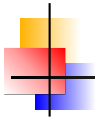
Decision making is viewed as deduction, an agent's program is encoded as a logical theory, and actions selection reduces to a problem of proof.

### Advantages:

- ▶ elegance and a clean (logical) semantics.

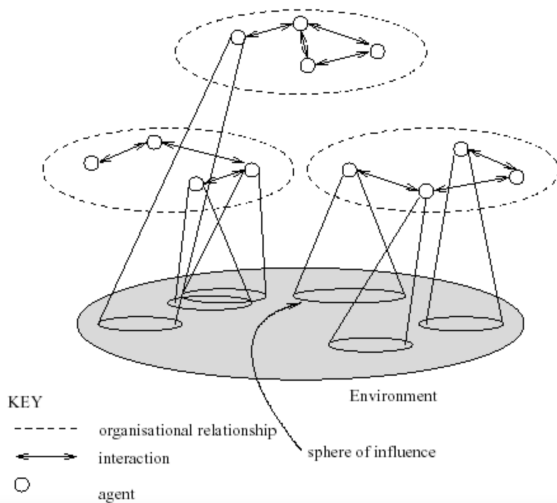
### Disadvantages:

- ▶ inherent computational complexity of theorem proving;
- ▶ based on the assumption of calculative rationality:
  - ▶ world will not change in any significant way while the agent is deciding what to do;
  - ▶ an action which is rational when decision-making begins will be rational when it concludes.
- ▶ transduction and representation/reasoning problems essentially unsolved.



# Multiagent interactions

## Multiagent interactions



*Figure 6:* Typical structure of a multiagent system



## *Multiagent interactions*

---

- ▶ The system contains **a number of agents**, which communicate with one another and act in an environment.
- ▶ Different agents have different **spheres of influence**; they have control over (or at least are able to influence) different parts of the environment.
- ▶ These spheres of influence may **intersect** or **coincide** in some cases. This fact gives rise to **dependencies** between the agents.
- ▶ Agents are typically linked by other **relationships**. For example, there might be 'power' relationships, where one agent is subordinate to another.

It is **critically important** to understand the type of **interaction** that takes place between the agents in order to be able to make the **best decision** possible about what **action** to perform.



We denote the set of agents by  $AG$ . Then

$$AG = \{1, \dots, n\}$$

for some  $n \geq 1$ . We use the notation  $i$  for an arbitrary agent.

- ▶ For every agent  $i$ , let  $Ac_i$  be the set of its **actions**.
- ▶ Agents are assumed to be **self-interested**: each agent has its own **preferences** about how the world should be.
- ▶ Assume there is a set  $\Omega$  of **outcomes** that agents have preferences over.
- ▶ Think of  $\Omega$  as the set of outcomes of a game that the agents are playing.



### Definition 1.39

The **utility function** of an agent  $i$  is a function  $u_i : \Omega \rightarrow \mathbb{R}$ .

- ▶ For every  $\omega \in \Omega$ ,  $u_i(\omega) \in \mathbb{R}$  indicates **how good** the outcome  $\omega$  is for  $i$ . The larger the number the better.

### Definition 1.40

The **preference ordering** of agent  $i$  is the binary relation  $\succeq_i$  on  $\Omega$  defined as follows: for every  $\omega, \omega' \in \Omega$ ,

$$\omega \succeq_i \omega' \text{ iff } u_i(\omega) \geq u_i(\omega').$$

- ▶ If  $\omega \succeq_i \omega'$ , we say that agent  $i$  **prefers**  $\omega$  over  $\omega'$  or that  $\omega$  **is preferred by** agent  $i$  over  $\omega'$ .



### Definition 1.41

The *strict preference ordering* of agent  $i$  is the binary relation  $\succ_i$  on  $\Omega$  defined as follows: for every  $\omega, \omega' \in \Omega$ ,

$$\omega \succ_i \omega' \text{ iff } u_i(\omega) > u_i(\omega').$$

- ▶ If  $\omega \succ_i \omega'$ , we say that agent  $i$  *strictly prefers*  $\omega$  over  $\omega'$  or that  $\omega$  *is strictly preferred by* agent  $i$  over  $\omega'$ .

### Proposition 1.42

- ▶  $\succeq_i$  is reflexive, transitive and total.
- ▶  $\succ_i$  is irreflexive and transitive.



- ▶ Utility functions are just a way of **representing an agent's preferences**.
- ▶ Utility  $\neq$  **money**.

### Example 1.43

Suppose that Alice has \$500 million in the bank, while Mary is absolutely penniless. A rich benefactor generously wishes to donate one million dollars to one of them.

- ▶ If the benefactor gives the money to Alice, there will be some increase in the utility of Alice's situation, but there is not much that you can do with \$501 million that you cannot do with \$500 million.
- ▶ In contrast, if the benefactor gives the money to Mary, the increase in Mary's utility is **enormous**: she goes from having no money at all to being a millionaire. That is a huge difference.



We introduce a model of the environment in which our agents act.

- ▶ The agents **simultaneously choose** an **action** to perform.
- ▶ As a result of their actions, an **outcome** in  $\Omega$  will result.
- ▶ The **actual** outcome depends on the **combination** of actions.
- ▶ The agents have no choice about whether to perform an action — they have to simply go ahead and perform one.
- ▶ An agent **cannot see** the action performed by the other agents.
- ▶ Environment behaviour is given by state transformer functions.



### Notation 1.44

Let us denote  $Ac_1 \times Ac_2 \times \dots \times Ac_n$  by  $Ac$ . Elements of  $Ac$  are also called **action profiles**.

### Definition 1.45

A function  $\tau : Ac \rightarrow \Omega$  is said to be a **state transformer** function.

### Definition 1.46

Let  $i$  be an agent. We define the binary relations  $\succeq_i, \succ_i$  on  $Ac$  as follows: for all  $a, a' \in Ac$ ,

$$a \succeq_i a' \quad \text{iff} \quad \tau(a) \succeq_i \tau(a') \quad \text{iff} \quad u_i(\tau(a)) \geq u_i(\tau(a')),$$

$$a \succ_i a' \quad \text{iff} \quad \tau(a) \succ_i \tau(a') \quad \text{iff} \quad u_i(\tau(a)) > u_i(\tau(a')).$$



## *Multiagent interactions - Example*

Assume that there are only two agents and that each agent has just two possible actions that it can perform: *C(cooperate)* and *D(defect)*. Thus,

$$AG = \{1, 2\}, \quad A_{c_1} = A_{c_2} = \{C, D\} \text{ and} \\ A_c = \{(D, D), (D, C), (C, D), (C, C)\}.$$

Let  $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4\}$  be the set of outcomes.

### *Example 1.47*

Define

$$\tau(D, D) = \omega_1, \quad \tau(D, C) = \omega_2, \quad \tau(C, D) = \omega_3, \quad \tau(C, C) = \omega_4. \quad (1)$$

$\tau$  maps each action profile to a *different* outcome. The environment is *sensitive* to the actions of *both* agents.

### Example 1.48

Define

$$\tau(D, D) = \omega_1, \tau(D, C) = \omega_1, \tau(C, D) = \omega_1, \tau(C, C) = \omega_1. \quad (2)$$

$\tau$  maps each action profile to the **same** outcome. It **does not matter** what the agents do: the outcome will be the same. Neither agent has any influence in such a scenario.

### Example 1.49

Define

$$\tau(D, D) = \omega_1, \tau(D, C) = \omega_2, \tau(C, D) = \omega_1, \tau(C, C) = \omega_2. \quad (3)$$

The environment is only **sensitive** to the actions performed by **one** of the agents, namely agent 2. It **does not matter** what agent 1 does.



## *Multiagent interactions - Example*

---

- ▶ Suppose we have the case where both agents can influence the outcome, that is the state transformer function is defined by (1).
- ▶ Define the utility functions as follows:

$$\begin{aligned}u_1(\omega_1) &= 1, & u_1(\omega_2) &= 1, & u_1(\omega_3) &= 4, & u_1(\omega_4) &= 4, \\u_2(\omega_1) &= 1, & u_2(\omega_2) &= 4, & u_2(\omega_3) &= 1, & u_2(\omega_4) &= 4.\end{aligned}$$

- ▶ With a bit of abuse of notation, we write

$$\begin{aligned}u_1(D, D) &= 1, & u_1(D, C) &= 1, & u_1(C, D) &= 4, & u_1(C, C) &= 4, \\u_2(D, D) &= 1, & u_2(D, C) &= 4, & u_2(C, D) &= 1, & u_2(C, C) &= 4.\end{aligned}$$



## *Multiagent interactions - Example*

---

- ▶ Agent 1's preferences are:

$$(C, C) \succeq_1 (C, D) \succ_1 (D, C) \succeq_1 (D, D).$$

- ▶ Consider the following question:

If you were agent 1 in this scenario, what would you choose to do — cooperate or defect?

- ▶ Agent 1 prefers all the outcomes in which it **cooperates** over all the outcomes in which it **defects**.
- ▶ The answer is clear: agent 1 should cooperate;  $C$  is the **rational choice** for agent 1.
- ▶ It does not matter, in this case, what agent 2 chooses to do.



## *Multiagent interactions - Example*

---

- ▶ Agent 2's preferences are:

$$(C, C) \succeq_2 (D, C) \succ_2 (C, D) \succeq_2 (D, D).$$

- ▶ The answer is again clear: agent 2 should cooperate;  $C$  is the **rational choice** for agent 2.
- ▶ In this scenario, the action one agent should perform does not depend in any way on what the other does.
- ▶ If both agents act **rationally**, then the **joint action** selected will be  $(C, C)$ : both agents will cooperate.





## *Multiagent interactions - Example*

- ▶ In the same environment, with the state transformer function defined by (1), assume that the utility functions are defined as follows:

$$\begin{aligned}u_1(D, D) &= 4, & u_1(D, C) &= 4, & u_1(C, D) &= 1, & u_1(C, C) &= 1, \\u_2(D, D) &= 4, & u_2(D, C) &= 1, & u_2(C, D) &= 4, & u_2(C, C) &= 1.\end{aligned}$$

- ▶ The two agents' preferences are:

$$\begin{aligned}(D, D) &\succeq_1 (D, C) \succ_1 (C, D) \succeq_1 (C, C), \\(D, D) &\succeq_2 (C, D) \succ_2 (D, C) \succeq_2 (C, C).\end{aligned}$$

- ▶ Both agents should **defect**. If they act **rationally**, then the **joint action** selected will be **(D, D)**.



## Pay-off matrix

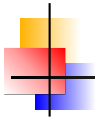
We can characterise the previous scenario in a **pay-off matrix**:

|   |   | 1        |          |
|---|---|----------|----------|
|   |   | D        | C        |
| 2 | D | 4      1 | 4      1 |
|   | C | 1      4 | 1      1 |

Corresponds to

$$\begin{aligned} u_1(D, D) &= 4, & u_1(D, C) &= 4, & u_1(C, D) &= 1, & u_1(C, C) &= 1, \\ u_2(D, D) &= 4, & u_2(D, C) &= 1, & u_2(C, D) &= 4, & u_2(C, C) &= 1. \end{aligned}$$

- ▶ This pay-off matrix in fact defines a **game**.
- ▶ Agent 1 is the **column player**. Agent 2 is the **row player**.
- ▶ Value in the **top right** of each cell = pay-off received by 1.
- ▶ Value in the **bottom left** of each cell = pay-off received by 2.



## *Solution Concepts and Solution Properties*

---

The basic question for each agent is:

What should I do?

We define some **solution concepts** and **solution properties**.



## *Solution Concepts and Solution Properties*

In the following, we assume that we have only **two agents**. Then  $AG = \{1, 2\}$ , there are two utility functions  $u_1, u_2 : \Omega \rightarrow \mathbb{R}$ ,  $Ac = Ac_1 \times Ac_2$  and  $\tau : Ac \rightarrow \Omega$  is the state transformer function.

We use terminology from game theory.

- ▶ We consider the **(two-agent) system** as a **(two-player) game**, where the **agents** are the **players**.
- ▶ **Actions** are called **strategies** and denoted by  $s, s', s_1, s_2$ , etc. Thus,  $s_i$  is a strategy of agent  $i$  means  $s_i \in Ac_i$ . We shall say that an agent  $i$  **plays a strategy**  $s_i$ .
- ▶ **Action profiles** are called **strategy profiles**.
- ▶ Thus, the pair  $(s_1, s_2)$  is a strategy profile iff  $s_1$  is a strategy of agent 1 and  $s_2$  is a strategy of agent 2.
- ▶ We write sometimes  $u_i(s_1, s_2)$  instead of  $u_i(\tau(s_1, s_2))$ .



### Definition 1.50

Let  $s_2$  be a strategy of agent 2. A strategy  $s_1$  of agent 1 is said to be its **best response** to  $s_2$  if

$$(s_1, s_2) \succeq_1 (s, s_2) \quad \text{for all } s \in Ac_1.$$

Thus, agent 1's **best response** to  $s_2$  is a strategy that gives agent 1 the highest pay-off when played against  $s_2$ .

### Definition 1.51

A **dominant strategy** for agent 1 is a strategy  $s_1$  that is the best response to **all** of agent 2's strategies.

Thus,  $s_1$  is a dominant strategy for agent 1 iff

$$(s_1, s') \succeq_1 (s, s') \quad \text{for all } s \in Ac_1 \text{ and all } s' \in Ac_2.$$

Similar definitions are given for agent 2.

### Definition 1.52

Let  $s_1$  be a strategy of agent 1. A strategy  $s_2$  of agent 2 is said to be its **best response** to  $s_1$  if

$$(s_1, s_2) \succeq_2 (s_1, s') \quad \text{for all } s' \in Ac_2.$$

Thus, agent 2's **best response** to  $s_1$  is a strategy that gives agent 2 the highest pay-off when played against  $s_1$ .

### Definition 1.53

A **dominant strategy** for agent 2 is a strategy  $s_2$  that is the best response to **all** of agent 1's strategies.

Thus,  $s_2$  is a dominant strategy for agent 2 iff

$$(s, s_2) \succeq_2 (s, s') \quad \text{for all } s \in Ac_1 \text{ and all } s' \in Ac_2.$$



## *Dominant strategies*

The presence of a dominant strategy for an agent makes the decision about what to do extremely easy:

the agent guarantees its best outcome by **performing** the dominant strategy.

Recall the example with the following **pay-off matrix**:

|   |   | 1   |     |
|---|---|-----|-----|
|   |   | D   | C   |
| 2 | D | 4 4 | 1 1 |
|   | C | 4 1 | 1 1 |

D (defect) is the dominant strategy for both agents.



### Definition 1.54

A strategy profile  $(s_1, s_2)$  is in **Nash equilibrium** if

- ▶  $s_1$  is agent 1's best response to  $s_2$ , and
- ▶  $s_2$  is agent 2's best response to  $s_1$ .

We also say that the two strategies  $s_1$  and  $s_2$  are in Nash equilibrium.

### Remark 1.55

A strategy profile  $(s_1, s_2)$  is in Nash equilibrium iff for all  $s \in Ac_1$  and all  $s' \in Ac_2$ ,

$$(s_1, s_2) \succeq_1 (s, s_2) \quad \text{and} \quad (s_1, s_2) \succeq_2 (s_1, s').$$





## Nash equilibrium

---

### *A reformulation*

A strategy profile  $(s_1, s_2)$  is in **Nash equilibrium** if

- ▶ under the assumption that agent 1 plays  $s_1$ , agent 2 can do no better than play  $s_2$ , and
- ▶ under the assumption that agent 2 plays  $s_2$ , agent 1 can do no better than play  $s_1$ .

Neither agent has any incentive to deviate from a Nash equilibrium.

### *Remark 1.56*

*If  $s_1$  is a dominant strategy for agent 1 and  $s_2$  is a dominant strategy for agent 2, then  $(s_1, s_2)$  is in Nash equilibrium.*



## *Nash equilibrium*

---

- ▶ Nash equilibrium was introduced by John Forbes Nash, Jr. in his 1950 PhD thesis.
- ▶ One of the most important concepts in game theory.
- ▶ One of the most important concepts in analysing multiagent systems.
- ▶ The type of Nash equilibrium defined above is known as **pure strategy Nash equilibrium**.
- ▶ If in a given scenario there exists a Nash equilibrium, then it is clear what to do.

Unfortunately,

- ▶ Not every interaction scenario has a Nash equilibrium.
- ▶ Some interaction scenarios have more than one Nash equilibrium.



## Matching Pennies

Players 1 and 2 simultaneously choose the face of a coin, either **heads** or **tails**. If they show the same face, then player 2 wins, while if they show different faces, then player 1 wins.

### Pay-off matrix

|   |       | 1         |           |
|---|-------|-----------|-----------|
|   |       | heads     | tails     |
| 2 | heads | 1      -1 | -1      1 |
|   | tails | -1      1 | 1      -1 |

**There is no Nash equilibrium:** no matter which strategy profile we choose, one of the agents would have preferred to have made another choice, under the assumption that the other player did not change its choice.



## *Rock-Paper-Scissors*

---

This game is played by two players, as follows.

- ▶ The players must simultaneously declare one of three possible choices: **rock**, **scissors**, or **paper**.
- ▶ They usually do this by showing a fist for rock, an open palm for paper, or a V-shape with the fingers for scissors.
- ▶ To determine the winner, the following rule is used:  
**Paper covers rock; scissors cut paper; rock blunts scissors.**
- ▶ In other words, if you declare paper and I declare rock, then you win; if you declare paper and I declare scissors, then I win; if I declare scissors and you declare rock, then you win, and so on.



## Rock-Paper-Scissors

### Pay-off matrix

|   |          | 1       |         |          |
|---|----------|---------|---------|----------|
|   |          | rock    | paper   | scissors |
| 2 | rock     | 0<br>0  | 1<br>-1 | -1<br>1  |
|   | paper    | -1<br>1 | 0<br>0  | 1<br>-1  |
|   | scissors | 1<br>-1 | -1<br>1 | 0<br>0   |

- ▶ **There is no dominant strategy:** whatever choice you make, you can end up with the worst possible outcome: a utility of  $-1$ .
- ▶ **There is no Nash equilibrium.**

### Definition 1.57

An outcome is **Pareto efficient** (or **Pareto optimal**) if there is no other outcome that improves one agent's utility without making somebody else worse off.

### Remark 1.58

An outcome is **Pareto inefficient** if there is another outcome that makes at least one agent better off without making anybody else worse off.

Let  $\omega$  be an outcome.

- ▶ If  $\omega$  is Pareto optimal, then at least one agent doesn't wish to move away from  $\omega$  (as this agent will be worse off).
- ▶ If  $\omega$  is **not** Pareto optimal, then there is another outcome  $\omega'$  that makes **everyone** as happy, if not happier than  $\omega$ .  
**Reasonable** agents would agree to move to  $\omega'$ .



- ▶ Pareto efficiency is not a solution concept, it is a **solution property**.
- ▶ Desirable solutions should satisfy it, but it is not very useful as a way of selecting outcomes.

### Example 1.59

Suppose a brother and sister are playing the game of **dividing a chocolate cake among themselves**.

- ▶ The outcome in which the **sister gets the whole cake** is Pareto efficient. However, it is hard to see this as a reasonable outcome.
- ▶ **Any way** of dividing a cake between the brother and sister in which the whole cake is given out will be Pareto optimal.



### Definition 1.60

The *social welfare* of an outcome  $\omega$ , denoted by  $sw(\omega)$ , is defined as follows

$$sw(\omega) = \sum_{i \in AG} u_i(\omega).$$

- ▶  $sw(\omega)$  measures how much utility is created by the outcome  $\omega$  in total.
- ▶ It is an important property of outcomes, but is not generally a way of directly selecting outcomes.





### Maximizing social welfare

- ▶ is not relevant from an individual agent's point of view.
- ▶ becomes **relevant** when the whole system (all agents) has a **single owner**. Then overall benefit of the system is important, not individuals.

### *Example 1.61*

Suppose you have a game with 100 players, in which one outcome gives 101 million to one player and nothing to the rest, while every other outcome gives 1 million to each.

The first outcome maximizes social welfare, although the other 99 players might not be very happy with such an outcome.



## Strictly competitive games

### Definition 1.62

A two-agent system (two-person game) is said to be **strictly competitive** if the following holds for all outcomes  $\omega, \omega'$ :

$$\omega \succ_1 \omega' \quad \text{iff} \quad \omega' \succ_2 \omega.$$

Thus, the preferences of the players are opposed to one another:  $\omega$  is strictly preferred by agent 1 over  $\omega'$  iff  $\omega'$  is strictly preferred by agent 2 over  $\omega$ .

- ▶ Chess and checkers are obvious examples of strictly competitive games.
- ▶ Any game in which the possible outcomes are win, lose or draw is strictly competitive.



## Zero-sum games

---

### Definition 1.63

A two-agent system (two-person game) is said to be **zero-sum** if the following holds for any outcome  $\omega$ :

$$u_1(\omega) + u_2(\omega) = 0.$$

There is no possibility of cooperative behaviour:

- ▶ The best outcome for you is the worst outcome for your opponent.
- ▶ If you allow your opponent to get **positive** utility, then you get **negative** utility.
- ▶ Matching Pennies and Rock-Paper-Scissors are zero-sum games.



## *Prisoner's Dilemma*

---

The following is a famous game-theoretic scenario.

Two men are collectively charged with a crime and held in separate cells. They have no way of communicating with each other or making any kind of agreement. The two men are told that:

1. If **one** of them **confesses** to the crime and the **other does not**, the **confessor** will be **freed**, and the **other** will be **jailed** for **three** years.
2. If **both confess** to the crime, then each will be **jailed** for **two** years.

Both prisoners know that if **neither confesses**, then **they** will each be **jailed** for **one** year.



## *Prisoner's Dilemma*

---

What should a prisoner do? The 'standard' approach to this problem is to put yourself in the place of a prisoner and reason as follows.

- ▶ Suppose the other prisoner confesses. Then my best response is to confess.
- ▶ Suppose the other prisoner does not confess. Then my best response is to confess.

In other words, confession is the best strategy for a prisoner. Hence, both prisoners should confess and, as a result, each of them will be **jailed** for **two** years.

But **this is not the best the prisoners can do**. Surely if neither of them confesses, then they would do better: each will be **jailed** for **one** year.



## Prisoner's Dilemma

- ▶ We refer to **confessing** as **D(defect)**, and **not confessing** as **C(cooperate)**.
- ▶ There are four possible outcomes, so the environment is of type (1).
- ▶ The associated pay-off matrix is

|   |   |   |   |
|---|---|---|---|
|   |   | 1 |   |
|   |   | D | C |
| 2 | D | 3 | 2 |
|   | C | 5 | 4 |

Thus,

$$\begin{aligned} u_1(D, D) &= 3, & u_1(D, C) &= 5, & u_1(C, D) &= 2, & u_1(C, C) &= 4, \\ u_2(D, D) &= 3, & u_2(D, C) &= 2, & u_2(C, D) &= 5, & u_2(C, C) &= 4. \end{aligned}$$



## Prisoner's Dilemma

---

The two players' preferences are:

$$(D, C) \succ_1 (C, C) \succ_1 (D, D) \succ_1 (C, D),$$

$$(C, D) \succ_2 (C, C) \succ_2 (D, D) \succ_2 (D, C).$$

- ▶  $D$ (defect) is a dominant strategy for both players. If they act **rationally**, then they should both defect.
- ▶  $(D, D)$  is the only Nash equilibrium.
- ▶ So, it seems that we are inevitably going to end up with the  $(D, D)$  outcome, with  $u_1(D, D) = u_2(D, D) = 3$ .



## *Prisoner's Dilemma*

---

- ▶ However, if they both cooperated, then one gets the  $(C, C)$  outcome, with better utilities  $u_1(C, C) = u_2(C, C) = 4$ .
- ▶  $(C, C)$  maximizes social welfare.
- ▶ All outcomes except  $(D, D)$  are Pareto optimal.

The fact that utility seems to be wasted here, and that the agents could both do better by cooperating, even though the rational thing to do is to defect, is why this is referred to as a **dilemma**.



The prisoner's dilemma may seem an abstract problem, but it turns out to be very common in the real world.

### *Example 1.64*

Consider the problem of **nuclear weapons treaty compliance**. Two countries 1 and 2 have signed a treaty to dispose of their nuclear weapons. Each country can then either cooperate (i.e. get rid of their weapons), or defect (i.e. keep their weapons).

- ▶ If you get rid of your weapons, you run the risk that the other side keeps theirs
- ▶ If you keep yours, then the possible outcomes are that you will have nuclear weapons while the other country does not, or else at worst that you both retain your weapons.
- ▶ This is not the best possible outcome, but is certainly better than you giving up your weapons while your opponent keeps theirs, which is what you risk if you give up your weapons.



# Logical foundations



### *Logics of knowledge and belief*

- ▶ were developed for reasoning about multiagent systems.
- ▶ are used to prove properties of these systems.
- ▶ are used to represent and reason about the **information** that agents possess: their **knowledge** and **belief**
- ▶ are based on **modal logic**.

We look at how one might represent statements such as

- ▶ “John knows that it is raining.”
- ▶ “John believes that it will rain tomorrow.” “Mary knows that John believes that it will rain tomorrow.”
- ▶ “It is common knowledge between Mary and John that it is raining.”
- ▶ “Janine believes Cronos is the father of Zeus.”



Consider the following statement:

Janine believes Cronos is the father of Zeus.

How can we formalize it in first-order logic?

An attempt is

$$\varphi := \text{Believe}(\text{Janine}, \text{Father}(\text{Zeus}, \text{Cronos})),$$

where *Believe* and *Father* are binary relation symbols, while *Janine*, *Zeus*, *Cronos* are constant symbols.

### *A syntactic problem*

$\varphi$  is not a formula of first-order logic, as *Father*(*Zeus*, *Cronos*) is a formula, not a term.



### A semantic problem

- ▶ Consider another constant symbol *Jupiter*, denoting the same individual as *Zeus*: the supreme deity of the classical world.
- ▶ It is therefore natural to accept as true the formula

$$\psi := (\textit{Zeus} = \textit{Jupiter}).$$

- ▶ Let us denote

$$\chi := \textit{Believe}(\textit{Janine}, \textit{Father}(\textit{Jupiter}, \textit{Cronos})),$$

- ▶ If  $\varphi, \chi$  were formulas of first-order logic, then, from the truth of  $\psi$ , we could infer that  $\varphi$  and  $\chi$  are equivalent.
- ▶ Intuition rejects this derivation as invalid: believing that the father of Zeus is Cronos **is not the same as** believing that the father of Jupiter is Cronos.



### *A semantic problem*

The problem is that, in general the truth value of the sentence  
Janine believes  $p$ .

does not depend solely on the truth value of  $p$ . In other words,  
**belief is not truth-functional.**

Similarly, **knowledge is not truth-functional.**

- ▶ First-order logic is not suitable in its standard form for reasoning about notions such as belief and knowledge.
- ▶ Alternative formalisms are required.

The field of **logics of knowledge and belief** has begun with the publication, in 1962, of Jaakko Hintikka's book **Knowledge and Belief. An Introduction to the Logic of the Two Notions.**



### *Approach to the syntactic problem:*

- ▶ Use **modal** logics, whose language contains non-truth-functional **modal operators**, which are applied to formulae.
- ▶ An example of such logics are **epistemic logics**.

### *Approach to the semantic problem:*

- ▶ Use a **possible-worlds semantics**, originally proposed for epistemic logic by Hintikka in the aforementioned book.
- ▶ An agent's beliefs, knowledge, goals, etc., are characterized in terms of a set of **possible worlds** (called **epistemic alternatives** by Hintikka), with an **accessibility** relation holding between them.
- ▶ Something true in **all** our agent's epistemic alternatives could be said to be believed by the agent.



The most common formulation of possible-worlds semantics in normal modal logics was developed by Kripke:

- ▶ Saul Kripke, *Semantical analysis of modal logic I. Normal modal propositional calculi*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, 9 (1963), 67-96.





# Modal Logics

Textbook:

P. Blackburn, M. de Rijke, Y. Venema, Modal logic, Cambridge Tracts in Theoretical Computer Science 53, Cambridge University Press, 2001



### Definition 1.65

The *basic modal language*  $ML_0$  consists of:

- ▶ a set  $PROP$  of *atomic propositions* (denoted  $p, q, r, v, \dots$ );
- ▶ the propositional connectives:  $\neg, \rightarrow$ ;
- ▶ the propositional constant  $\perp$  (*false*);
- ▶ parentheses:  $(, )$ ;
- ▶ the modal operator  $\Box$  (*box*).

The set  $Sym(ML_0)$  of *symbols* of  $ML_0$  is

$$Sym(ML_0) := PROP \cup \{\neg, \rightarrow, \perp, (, ), \Box\}.$$

The *expressions* of  $ML_0$  are the finite sequences of symbols of  $ML_0$ .



### Definition 1.66

The **formulas** of the basic modal language  $ML_0$  are the expressions inductively defined as follows:

- (F0) Every atomic proposition is a formula.
- (F1)  $\perp$  is a formula.
- (F2) If  $\varphi$  is a formula, then  $(\neg\varphi)$  is a formula.
- (F3) If  $\varphi$  and  $\psi$  are formulas, then  $(\varphi \rightarrow \psi)$  is a formula.
- (F4) If  $\varphi$  is a formula, then  $(\Box\varphi)$  is a formula.
- (F5) Only the expressions obtained by applying rules (F0), (F1), (F2), (F3), (F4) are formulas.

**Notation:** The set of formulas is denoted by  $Form(ML_0)$ .



## Basic modal language

Formulas of  $ML_0$  are defined, using the Backus-Naur notation, as follows:

$$\varphi ::= p \mid \perp \mid (\neg\varphi) \mid (\varphi \rightarrow \psi) \mid (\Box\varphi), \quad \text{where } p \in PROP.$$

### Derived connectives

Connectives  $\vee$ ,  $\wedge$ ,  $\leftrightarrow$  and the constant  $\top$  (**true**) are introduced as in classical propositional logic:

$$\begin{aligned}\varphi \vee \psi &:= ((\neg\varphi) \rightarrow \psi) & \varphi \wedge \psi &:= \neg(\varphi \rightarrow (\neg\psi)) \\ \varphi \leftrightarrow \psi &:= ((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)) & \top &:= \neg\perp.\end{aligned}$$

### Dual modal operator

The dual of  $\Box$  is denoted by  $\Diamond$  (**diamond**) and is defined as:

$$\Diamond\varphi := \neg\Box\neg\varphi$$

for every formula  $\varphi$ .



Usually the external parantheses are omitted, we write them only when necessary. We write  $\neg\varphi, \varphi \rightarrow \psi, \Box\varphi$ .

To reduce the use of parentheses, we assume that

- ▶ modal operators  $\Diamond$  and  $\Box$  have higher precedence than the other connectives.
- ▶  $\neg$  has higher precedence than  $\rightarrow, \wedge, \vee, \leftrightarrow$ ;
- ▶  $\wedge, \vee$  have higher precedence than  $\rightarrow, \leftrightarrow$ .



### Classical modal logic

In classical modal logic,  $\Diamond\varphi$  is read as **it is possible the case that  $\varphi$** . Then  $\Box\varphi$  means **it is not possible that not  $\varphi$** , that is **necessarily  $\varphi$** .

Examples of formulas we would probably regard as correct principles

- ▶  $\Box\varphi \rightarrow \Diamond\varphi$  (**whatever is necessary is possible**)
- ▶  $\varphi \rightarrow \Diamond\varphi$  (**whatever is, is possible**).

The status of other formulas is harder to decide. What can we say about  $\varphi \rightarrow \Box\Diamond\varphi$  (**whatever is, is necessarily possible**) or  $\Diamond\varphi \rightarrow \Box\Diamond\varphi$  (**whatever is possible, is necessarily possible**)? Can we consider them as general truths? In order to give an answer to such questions, one has to define a **semantics** for the classical modal logic.



### Definition 1.67

A **relational structure** is a tuple  $\mathcal{F}$  consisting of:

- ▶ a nonempty set  $W$ , called the **universe** (or **domain**) of  $\mathcal{F}$ , and
- ▶ a set of relations on  $W$ .

We assume that every relational structure contains at least one relation. The elements of  $W$  are called **points**, **nodes**, **states**, **worlds**, **times**, **instances** or **situations**.

### Example 1.68

A partially ordered set  $\mathcal{F} = (W, R)$ , where  $R$  is a partial order relation on  $W$ .



**Labeled Transition Systems (LTSs)**, or more simply, transition systems, are very simple relational structures widely used in computer science.

### *Definition 1.69*

An **LTS** is a pair  $(W, \{R_a \mid a \in A\})$ , where  $W$  is a nonempty set of **states**,  $A$  is a nonempty set of **labels** and, for every  $a \in A$ ,

$$R_a \subseteq W \times W$$

is a binary relation on  $W$ .

LTSs can be viewed as an abstract model of computation: the states are the possible states of a computer, the labels stand for programs, and  $(u, v) \in R_a$  means that there is an execution of the program  $a$  starting in state  $u$  and terminating in state  $v$ .





Let  $W$  be a nonempty set and  $R \subseteq W \times W$  be a binary relation.

We write usually  $Rwv$  instead of  $(w, v) \in R$ . If  $Rwv$ , then we say that  $v$  is  **$R$ -accessible** from  $w$ .

The **inverse** of  $R$ , denoted by  $R^{-1}$ , is defined as follows:

$$R^{-1}vw \text{ iff } Rwv.$$

We define  $R^n$  ( $n \geq 0$ ) inductively:

$$R^0 = \{(w, w) \mid w \in W\}, \quad R^1 = R, \quad R^{n+1} = R \circ R^n.$$

Thus, for any  $n \geq 2$ , we have that  $R^n wv$  iff there exists  $u_1, \dots, u_{n-1}$  such that  $Rwu_1, Ru_1u_2, \dots, Ru_{n-1}v$ .



In the sequel we give the **semantics** of the basic modal language  $ML_0$ .

We will do this in two distinct ways:

- ▶ at the level of **models**, where the fundamental notion of **satisfaction** (or **truth**) is defined.
- ▶ at the level of frames, where the key notion of **validity** is defined.

### *Definition 1.70*

A **frame** for  $ML_0$  is a pair  $\mathcal{F} = (W, R)$  such that

- ▶  $W$  is a nonempty set;
- ▶  $R$  is a binary relation on  $W$ .

That is, a frame for the basic modal language is simply a relational structure with a single binary relation.

### Definition 1.71

A **model** for  $ML_0$  is a pair  $\mathcal{M} = (\mathcal{F}, V)$ , where

- ▶  $\mathcal{F} = (W, R)$  is a frame for  $ML_0$ ;
- ▶  $V : PROP \rightarrow 2^W$  is a function called **valuation**.

Thus,  $V$  assigns to each atomic proposition  $p \in PROP$  a subset  $V(p)$  of  $W$ . Informally, we think of  $V(p)$  as the set of points in the model  $\mathcal{M}$  where  $p$  is true.

Note that models for  $ML_0$  can also be viewed as relational structures in a natural way:

$$\mathcal{M} = (W, R, \{V(p) \mid p \in PROP\}).$$

Thus, a model is a relational structure consisting of a domain, a single binary relation  $R$  and the unary relations  $V(p), p \in PROP$ . A frame  $\mathcal{F}$  and a model  $\mathcal{M}$  are two relational structures based on the same universe. However, as we shall see, frames and models are used **very** differently.



## Frames and models

---

Let  $\mathcal{F} = (W, R)$  be a frame and  $\mathcal{M} = (\mathcal{F}, V)$  be a model. We also write  $\mathcal{M} = (W, R, V)$ .

We say that the model  $\mathcal{M} = (\mathcal{F}, V)$  is **based on** the frame  $\mathcal{F} = (W, R)$  or that  $\mathcal{F}$  is the frame **underlying**  $\mathcal{M}$ . Elements of  $W$  are called **states** in  $\mathcal{F}$  or in  $\mathcal{M}$ . We often write  $w \in \mathcal{F}$  or  $w \in \mathcal{M}$ .

### Remark

Elements of  $W$  are also called **worlds** or **possible worlds**, having as inspiration Leibniz's philosophy and the reading of basic modal language in which

$\Diamond\varphi$  means **possibly  $\varphi$**  and  $\Box\varphi$  means **necessarily  $\varphi$** .

In Leibniz's view, **necessity** means **truth in all possible worlds** and **possibility** means **truth in some possible world**.



We define now the notion of satisfaction.

### Definition 1.72

Let  $\mathcal{M} = (W, R, V)$  be a model and  $w$  a state in  $\mathcal{M}$ . We define inductively the notion

formula  $\varphi$  *is satisfied (or true) in  $\mathcal{M}$  at state  $w$ ,*

*Notation*  $\mathcal{M}, w \Vdash \varphi$

$\mathcal{M}, w \Vdash p$  iff  $w \in V(p)$ , where  $p \in PROP$

$\mathcal{M}, w \Vdash \perp$  never

$\mathcal{M}, w \Vdash \neg\varphi$  iff it is not true that  $\mathcal{M}, w \Vdash \varphi$

$\mathcal{M}, w \Vdash \varphi \rightarrow \psi$  iff  $\mathcal{M}, w \Vdash \varphi$  implies  $\mathcal{M}, w \Vdash \psi$

$\mathcal{M}, w \Vdash \Box\varphi$  iff for every  $v \in W$ ,  $Rwv$  implies  $\mathcal{M}, v \Vdash \varphi$ .

Let  $\mathcal{M} = (W, R, V)$  be a model.

### Notation

If  $\mathcal{M}$  does not satisfy  $\varphi$  at  $w$ , we write  $\mathcal{M}, w \not\models \varphi$  and we say that  $\varphi$  is **false** in  $\mathcal{M}$  at state  $w$ .

It follows from Definition 1.72 that for every state  $w \in W$ ,

- ▶  $\mathcal{M}, w \not\models \perp$
- ▶  $\mathcal{M}, w \models \neg\varphi$  iff  $\mathcal{M}, w \not\models \varphi$ .

### Notation

We can extend the valuation  $V$  from atomic propositions to arbitrary formulas  $\varphi$  so that  $V(\varphi)$  is the set of all states in  $\mathcal{M}$  at which  $\varphi$  is true:

$$V(\varphi) = \{w \mid \mathcal{M}, w \models \varphi\}.$$



Let  $\mathcal{M} = (W, R, V)$  be a model and  $w$  a state in  $\mathcal{M}$ .

### Proposition 1.73

For every formulas  $\varphi, \psi$ ,

$\mathcal{M}, w \Vdash \varphi \vee \psi$  iff  $\mathcal{M}, w \Vdash \varphi$  or  $\mathcal{M}, w \Vdash \psi$

$\mathcal{M}, w \Vdash \varphi \wedge \psi$  iff  $\mathcal{M}, w \Vdash \varphi$  and  $\mathcal{M}, w \Vdash \psi$

### Proposition 1.74

For every formula  $\varphi$ ,

$\mathcal{M}, w \Vdash \Diamond\varphi$  iff there exists  $v \in W$  such that  $Rwv$  and  $\mathcal{M}, v \Vdash \varphi$ .



Let  $\mathcal{M} = (W, R, V)$  be a model and  $w$  a state in  $\mathcal{M}$ .

### Proposition 1.75

For every  $n \geq 1$  and every formula  $\varphi$ , define

$$\Diamond^n \varphi := \underbrace{\Diamond \Diamond \dots \Diamond}_{n \text{ times}} \varphi, \quad \Box^n \varphi := \underbrace{\Box \Box \dots \Box}_{n \text{ times}} \varphi.$$

Then

$\mathcal{M}, w \Vdash \Diamond^n \varphi$  iff there exists  $v \in V$  s.t.  $R^n wv$  and  $\mathcal{M}, v \Vdash \varphi$   
 $\mathcal{M}, w \Vdash \Box^n \varphi$  iff for every  $v \in V$ ,  $R^n wv$  implies  $\mathcal{M}, v \Vdash \varphi$ .



Let  $\mathcal{M} = (W, R, V)$  be a model.

### Definition 1.76

- ▶ A formula  $\varphi$  is **globally true** or simply **true** in  $\mathcal{M}$  if  $\mathcal{M}, w \Vdash \varphi$  for every  $w \in W$ . **Notation:**  $\mathcal{M} \Vdash \varphi$
- ▶ A formula  $\varphi$  is **satisfiable** in  $\mathcal{M}$  if there exists a state  $w \in W$  such that  $\mathcal{M}, w \Vdash \varphi$ .

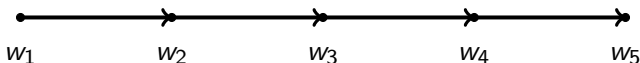
### Definition 1.77

Let  $\Sigma$  be a set of formulas.

- ▶  $\Sigma$  is **true** at state  $w$  in  $\mathcal{M}$  if  $\mathcal{M}, w \Vdash \varphi$  for every  $\varphi \in \Sigma$ .  
**Notation:**  $\mathcal{M}, w \Vdash \Sigma$
- ▶  $\Sigma$  is **globally true** or simply **true** in  $\mathcal{M}$  if  $\mathcal{M}, w \Vdash \Sigma$  for every state  $w$  in  $\mathcal{M}$ . **Notation:**  $\mathcal{M} \Vdash \Sigma$
- ▶  $\Sigma$  is **satisfiable** in  $\mathcal{M}$  if there exists a state  $w \in W$  such that  $\mathcal{M}, w \Vdash \Sigma$ .

## Example 1.78

Consider the frame  $\mathcal{F} = (W = \{w_1, w_2, w_3, w_4, w_5\}, R)$ , where  $Rw_iw_j$  iff  $j = i + 1$ :



Let us choose a valuation  $V$  such that  $V(p) = \{w_2, w_3\}$ ,  $V(q) = \{w_1, w_2, w_3, w_4, w_5\}$  and  $V(r) = \emptyset$ .

Consider the model  $\mathcal{M} = (\mathcal{F}, V)$ . Then

- (i)  $\mathcal{M}, w_1 \Vdash \Diamond \Box p$
- (ii)  $\mathcal{M}, w_1 \not\Vdash \Diamond \Box p \rightarrow p$
- (iii)  $\mathcal{M}, w_2 \Vdash \Diamond(p \wedge \neg r)$
- (iv)  $\mathcal{M}, w_1 \Vdash q \wedge \Diamond(q \wedge \Diamond(q \wedge \Diamond(q \wedge \Diamond q)))$
- (v)  $\mathcal{M} \Vdash \Box q$ .



The notion of satisfaction is **internal** and **local**. We evaluate formulas **inside** models, at some particular state  $w$  (the **current state**). Modal operators  $\Diamond, \Box$  work locally: we verify the truth of  $\varphi$  **only** in the states that are  $R$ -accessible from the current one.

At first sight this may seem a weakness of the satisfaction definition. In fact, it is its greatest source of strength, as it gives us great flexibility.

For example, if we take  $R = W \times W$ , then all states are accessible from the current state; this corresponds to the Leibnizian idea in its purest form.

Going to the other extreme, if we take  $R = \{(v, v) \mid v \in W\}$ , then no state has access to any other.

Between these extremes there is a wide range of options to explore.



We can ask ourselves the following natural questions:

- ▶ What happens if we impose some conditions on  $R$  (for example, reflexivity, symmetry, transitivity, etc.)?
- ▶ What is the impact of these conditions on the notions of necessity and possibility?
- ▶ What principles or rules are justified by these conditions?



Validity in a frame is one of the key concepts in modal logic.

### *Definition 1.79*

*Let  $\mathcal{F}$  be a frame and  $\varphi$  be a formula.*

- ▶  $\varphi$  is **valid at a state**  $w$  in  $\mathcal{F}$  if  $\varphi$  is true at  $w$  in every model  $\mathcal{M} = (\mathcal{F}, V)$  based on  $\mathcal{F}$ .
- ▶  $\varphi$  is **valid in**  $\mathcal{F}$  if it is valid at every state  $w$  in  $\mathcal{F}$ .

*Notation:*  $\mathcal{F} \Vdash \varphi$

Hence, a formula is valid in a frame if it is true at every state in every model based on the frame.



Validity in a frame differs in an essential way from the truth in a model. Let us give a simple example.

### *Example 1.80*

If  $\varphi \vee \psi$  is true in a model  $\mathcal{M}$  at  $w$ , then  $\varphi$  is true in  $\mathcal{M}$  at  $w$  or  $\psi$  is true in  $\mathcal{M}$  at  $w$  (by Proposition 1.73).

On the other hand, if  $\varphi \vee \psi$  is valid in a frame  $\mathcal{F}$  at  $w$ , it does not follow that  $\varphi$  is valid in  $\mathcal{F}$  at  $w$  or  $\psi$  is valid in  $\mathcal{F}$  at  $w$  ( $p \vee \neg p$  is a counterexample).



### Definition 1.81

Let  $\mathbf{M}$  be a class of models,  $\mathbf{F}$  be a class of frames and  $\varphi$  be a formula. We say that

- ▶  $\varphi$  is **true in  $\mathbf{M}$**  if it is true in every model in  $\mathbf{M}$ .

*Notation:*  $\mathbf{M} \models \varphi$

- ▶  $\varphi$  is **valid in  $\mathbf{F}$**  if it is valid in every frame in  $\mathbf{F}$ .

*Notation:*  $\mathbf{F} \models \varphi$

### Definition 1.82

The set of all formulas of  $ML_0$  that are valid in a class of frames  $\mathbf{F}$  is called the **logic of  $\mathbf{F}$**  and is denoted by  $\Lambda_{\mathbf{F}}$ .



### Example 1.83

Formulas  $\Diamond(\varphi \vee \psi) \rightarrow (\Diamond\varphi \vee \Diamond\psi)$  and  $\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$  are valid in the class of all frames.

### Example 1.84

Formula  $\Box\varphi \rightarrow \Box\Box\varphi$  is not valid in the class of all frames.

### Definition 1.85

We say that a frame  $\mathcal{F} = (W, R)$  is *transitive* if  $R$  is transitive.

### Example 1.86

Formula  $\Box\varphi \rightarrow \Box\Box\varphi$  is valid in the class of all transitive frames.





We introduce the **consequence relation**.

The basic ideas are the following;

- ▶ A relation of semantic consequence holds when the truth of the premises guarantees the truth of the conclusion.
- ▶ The inferences depend on the class of structures we are working with. (For example, inferences for transitive frames must be different than the ones for intransitive frames.)

Thus, the definition of the consequence relation must make reference to a class of structures  **$\mathcal{S}$** .

Let  $\mathbf{S}$  be a class of **structures** (frames or models) for  $ML_0$ .

If  $\mathbf{S}$  is a class of models, then a model **from**  $\mathbf{S}$  is simply an element  $\mathcal{M}$  of  $\mathbf{S}$ . If  $\mathbf{S}$  is a class of frames, then a model **from**  $\mathbf{S}$  is a model based on a frame in  $\mathbf{S}$ .

### *Definition 1.87*

Let  $\Sigma$  be a set of formulas and  $\varphi$  be a formula. We say that  $\varphi$  is a **semantic consequence of  $\Sigma$  over  $\mathbf{S}$**  if for all models  $\mathcal{M}$  from  $\mathbf{S}$  and all states  $w$  in  $\mathcal{M}$ ,

$$\mathcal{M}, w \Vdash \Sigma \quad \text{implies} \quad \mathcal{M}, w \Vdash \varphi.$$

*Notation:*  $\Sigma \Vdash_{\mathbf{S}} \varphi$

Thus, if  $\Sigma$  is true at a state of the model, then  $\varphi$  must be true **at the same state**.



### Remark 1.88

$$\{\psi\} \Vdash_{\mathbf{S}} \varphi \text{ iff } \mathbf{S} \Vdash \psi \rightarrow \varphi.$$

### Example 1.89

Let *Tran* be the class of transitive frames. Then

$$\{\Box\varphi\} \Vdash_{Tran} \Box\Box\varphi.$$

But  $\Box\Box\varphi$  is **NOT** a semantic consequence of  $\Box\varphi$  over the class of **all** frames.



### Definition 1.90

A **normal modal logic** is a set  $\Lambda$  of formulas of  $ML_0$  satisfying the following properties:

- ▶  $\Lambda$  contains the following **axioms**:

(Taut) all propositional tautologies,

(K)  $\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$ ,

where  $\varphi, \psi$  are formulas of  $ML_0$ .

- ▶  $\Lambda$  is closed under the following deduction rules:

▶ **modus ponens (MP)**: 
$$\frac{\varphi, \varphi \rightarrow \psi}{\psi}.$$

Hence, if  $\varphi \in \Lambda$  and  $\varphi \rightarrow \psi \in \Lambda$ , then  $\psi \in \Lambda$ .

▶ **generalization or necessitation**: 
$$\frac{\varphi}{\Box\varphi}.$$

Hence, if  $\varphi \in \Lambda$ , then  $\Box\varphi \in \Lambda$ .

We add all propositional tautologies as axioms for simplicity, it is not necessary. We could add a small number of tautologies, which generates all of them. For example,

$$(A1) \quad \varphi \rightarrow (\psi \rightarrow \varphi)$$

$$(A2) \quad (\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$$

$$(A3) \quad (\neg\psi \rightarrow \neg\varphi) \rightarrow (\varphi \rightarrow \psi).$$

### *Proposition 1.91*

*Any propositional tautology is valid in the class of all frames for  $ML_0$ .*

### *Remark 1.92*

*Tautologies may contain modalities, too. For example,  $\Diamond\psi \vee \neg\Diamond\psi$  is a tautology, since it has the same form as  $\varphi \vee \neg\varphi$ .*



## Normal modal logics - axiom (K)

---

Axiom (K) is sometimes called the **distribution axiom** and it is important because it allows us to transform  $\Box(\varphi \rightarrow \psi)$  (a boxed formula) in an implication  $\Box\varphi \rightarrow \Box\psi$ , enabling further pure propositional reasoning to take place.

For example, assume that we want to prove  $\Box\psi$  and we already have a proof that contains both  $\Box(\varphi \rightarrow \psi)$  and  $\Box\varphi$ . Applying (K) and modus ponens, we get  $\Box\varphi \rightarrow \Box\psi$ . Applying again modus ponens, we obtain  $\Box\psi$ .

By Example 1.83,

### *Proposition 1.93*

*(K) is valid in the class of all frames for  $ML_0$ .*



### Theorem 1.94

For any class  $\mathbf{F}$  of frames,  $\Lambda_{\mathbf{F}}$ , the logic of  $\mathbf{F}$ , is a normal modal logic.

### Lemma 1.95

- ▶ The collection of all formulas is a normal modal logic, called the *inconsistent logic*.
- ▶ If  $\{\Lambda_i \mid i \in I\}$  is a collection of normal modal logics, then  $\bigcap_{i \in I} \Lambda_i$  is a normal modal logic.

### Definition 1.96

$\mathbf{K}$  is the intersection of all normal modal logics.

Hence,  $\mathbf{K}$  is the smallest normal modal logic.



### Definition 1.97

A **K-proof** is a sequence of formulas  $\theta_1, \dots, \theta_n$  such that for any  $i \in \{1, \dots, n\}$ , one of the following conditions is satisfied:

- ▶  $\theta_i$  is an axiom (that is, a tautology or  $(K)$ );
- ▶  $\theta_i$  is obtained from previous formulas by applying modus ponens or generalization.

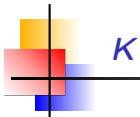
### Definition 1.98

Let  $\varphi$  be a formula. A **K-proof** of  $\varphi$  is a **K-proof**  $\theta_1, \dots, \theta_n$  such that  $\theta_n = \varphi$ .

If  $\varphi$  has a **K-proof**, we say that  $\varphi$  is **K-provable**.

**Notation:**  $\vdash_K \varphi$ .





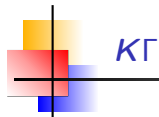
### *Theorem 1.99*

$$\mathbf{K} = \{\varphi \mid \vdash_{\mathbf{K}} \varphi\}.$$

The logic  $\mathbf{K}$  is very weak. If we are interested in transitive frames, we would like a proof system which reflects this. For example, we know that  $\Box\varphi \rightarrow \Box\Box\varphi$  is valid in the class of all transitive frames, so we would want a proof system that generates this formula.

$\mathbf{K}$  does not do this, since  $\Box\varphi \rightarrow \Box\Box\varphi$  is not valid in the class of all frames.

The idea is to extend  $\mathbf{K}$  with additional axioms.



By Lemma 1.95, for any set  $\Gamma$  of formulas, there exists the smallest normal modal logic that contains  $\Gamma$ .

### Definition 1.100

$K\Gamma$  is the smallest normal modal logic that contains  $\Gamma$ . We say that  $K\Gamma$  is *generated* by  $\Gamma$  or *axiomatized* by  $\Gamma$ .

### Definition 1.101

A  *$K\Gamma$ -proof* is a sequence of formulas  $\theta_1, \dots, \theta_n$  such that for any  $i \in \{1, \dots, n\}$ , one of the following conditions is satisfied:

- ▶  $\theta_i$  is an axiom (that is, a tautology or  $(K)$ );
- ▶  $\theta_i \in \Gamma$ ;
- ▶  $\theta_i$  is obtained from previous formulas by applying modus ponens or generalization.

### Definition 1.102

Let  $\varphi$  be a formula. A  **$K\Gamma$ -proof** of  $\varphi$  is a  $K\Gamma$ -proof  $\theta_1, \dots, \theta_n$  such that  $\theta_n = \varphi$ .

If  $\varphi$  has a  $K\Gamma$ -proof, we say that  $\varphi$  is  **$K\Gamma$ -provable**.

**Notation:**  $\vdash_{K\Gamma} \varphi$ .

### Theorem 1.103

$$K\Gamma = \{\varphi \mid \vdash_{K\Gamma} \varphi\}.$$



Let  $\Lambda$  be a normal modal logic.

### *Definition 1.104*

If  $\varphi \in \Lambda$ , we also say that  $\varphi$  is a  $\Lambda$ -theorem or a theorem of  $\Lambda$  and write  $\vdash_{\Lambda} \varphi$ . If  $\varphi \notin \Lambda$ , we write  $\not\vdash_{\Lambda} \varphi$ .

With these notations, the conditions from the definition of a normal modal logic are written as follows:

For any formulas  $\varphi, \psi$ , the following hold:

- (i) If  $\varphi$  is a tautology, then  $\vdash_{\Lambda} \varphi$ .
- (ii)  $\vdash_{\Lambda} (K)$ .
- (iii) If  $\vdash_{\Lambda} \varphi$  and  $\vdash_{\Lambda} \varphi \rightarrow \psi$ , then  $\vdash_{\Lambda} \psi$ .
- (iv) If  $\vdash_{\Lambda} \varphi$ , then  $\vdash_{\Lambda} \Box\varphi$ .



### Definition 1.105

Let  $\psi_1, \dots, \psi_n, \varphi$  be formulas. We say that  $\varphi$  is *deducible in propositional logic from assumptions*  $\psi_1, \dots, \psi_n$  if

$(\psi_1 \wedge \dots \wedge \psi_n) \rightarrow \varphi$  is a tautology.

### Proposition 1.106

$\Lambda$  is closed under propositional deduction: if  $\varphi$  is deducible in propositional logic from assumptions  $\psi_1, \dots, \psi_n$ , then

$\vdash_{\Lambda} \psi_1, \dots, \vdash_{\Lambda} \psi_n$  implies  $\vdash_{\Lambda} \varphi$ .

### Definition 1.107

Let  $\Gamma \cup \{\varphi\}$  be a set of formulas. We say that  $\varphi$  is **deducible in  $\Lambda$  from  $\Gamma$**  or that  $\varphi$  is  **$\Lambda$ -deducible from  $\Gamma$**  if there exist formulas  $\psi_1, \dots, \psi_n \in \Gamma$  ( $n \geq 0$ ) such that

$$\vdash_{\Lambda} (\psi_1 \wedge \dots \wedge \psi_n) \rightarrow \varphi.$$

(When  $n = 0$ , this means that  $\vdash_{\Lambda} \varphi$ ).

**Notation:**  $\Gamma \vdash_{\Lambda} \varphi$  We write  $\Gamma \not\vdash_{\Lambda} \varphi$  if  $\varphi$  is not  $\Lambda$ -deducible from  $\Gamma$ .

### Proposition 1.108 (Basic properties)

Let  $\varphi$  be a formula and  $\Gamma, \Delta$  be sets of formulas.

- (i)  $\emptyset \vdash_{\Lambda} \varphi$  iff  $\vdash_{\Lambda} \varphi$ .
- (ii)  $\vdash_{\Lambda} \varphi$  implies  $\Gamma \vdash_{\Lambda} \varphi$ .
- (iii)  $\varphi \in \Gamma$  implies  $\Gamma \vdash_{\Lambda} \varphi$ .
- (iv) If  $\Gamma \vdash_{\Lambda} \varphi$  and  $\Gamma \subseteq \Delta$ , then  $\Delta \vdash_{\Lambda} \varphi$ .

Let  $\varphi, \psi$  be formulas and  $\Gamma$  be a set of formulas,

### *Proposition 1.109*

$\Gamma \vdash_{\Lambda} \varphi$  iff there exists a finite subset  $\Sigma$  of  $\Gamma$  such that  $\Sigma \vdash_{\Lambda} \varphi$ .

### *Proposition 1.110*

- (i) If  $\Gamma \vdash_{\Lambda} \varphi$  and  $\psi$  is deducible in propositional logic from  $\varphi$ , then  $\Gamma \vdash_{\Lambda} \psi$ .
- (ii) If  $\Gamma \vdash_{\Lambda} \varphi$  and  $\Gamma \vdash_{\Lambda} \varphi \rightarrow \psi$ , then  $\Gamma \vdash_{\Lambda} \psi$ .
- (iii) If  $\Gamma \vdash_{\Lambda} \varphi$  and  $\{\varphi\} \vdash_{\Lambda} \psi$ , then  $\Gamma \vdash_{\Lambda} \psi$ .

### *Proposition 1.111 (Deduction Theorem)*

For any set of formulas  $\Gamma$  and any formulas  $\varphi, \psi$ ,

$$\Gamma \vdash_{\Lambda} \varphi \rightarrow \psi \quad \text{iff} \quad \Gamma \cup \{\varphi\} \vdash_{\Lambda} \psi.$$

### Definition 1.112

A set  $\Gamma$  of formulas is called  **$\Lambda$ -consistent** if  $\Gamma \not\vdash_{\Lambda} \perp$ .

If  $\Gamma$  is not  $\Lambda$ -consistent, we say that  $\Gamma$  is  **$\Lambda$ -inconsistent**.

A formula  $\varphi$  is  $\Lambda$ -consistent if  $\{\varphi\}$  is; otherwise, it is called  $\Lambda$ -inconsistent.

### Proposition 1.113

Let  $\Gamma$  be a set of formulas. The following are equivalent:

- (i)  $\Gamma$  is  $\Lambda$ -inconsistent.
- (ii) There exists a formula  $\psi$  such that  $\Gamma \vdash_{\Lambda} \psi$  and  $\Gamma \vdash_{\Lambda} \neg\psi$ .
- (iii)  $\Gamma \vdash_{\Lambda} \varphi$  for any formula  $\varphi$ .

### Proposition 1.114

$\Gamma$  is  $\Lambda$ -consistent iff any finite subset of  $\Gamma$  is  $\Lambda$ -consistent.



In the following, we say “normal logic ” instead of “normal modal logic”.

Let  $\mathbf{S}$  be a class of **structures** (frames or models) for  $ML_0$ .

*Notation:*

$$\Lambda_{\mathbf{S}} := \{\varphi \mid \mathcal{S} \Vdash \varphi \text{ for any structure } \mathcal{S} \text{ from } \mathbf{S}\}.$$

*Definition 1.115*

A normal logic  $\Lambda$  is **sound** with respect to  $\mathbf{S}$  if  $\Lambda \subseteq \Lambda_{\mathbf{S}}$ .

Thus,  $\Lambda$  is sound with respect to  $\mathbf{S}$  iff for any formula  $\varphi$  and for any structure  $\mathcal{S}$  in  $\mathbf{S}$ ,

$$\vdash_{\Lambda} \varphi \quad \text{implies} \quad \mathcal{S} \Vdash \varphi.$$

If  $\Lambda$  is sound with respect to  $\mathbf{S}$ , we say also that  $\mathbf{S}$  is a **class of frames (or models) for  $\Lambda$** .



### Definition 1.116

A normal logic  $\Lambda$  is

(i) **strongly complete** with respect to  $\mathbf{S}$  if for any set of formulas  $\Gamma \cup \{\varphi\}$ ,

$$\Gamma \Vdash_{\mathbf{S}} \varphi \text{ implies } \Gamma \vdash_{\Lambda} \varphi.$$

(ii) **weakly complete** with respect to  $\mathbf{S}$  if for any formula  $\varphi$ ,

$$\mathbf{S} \Vdash \varphi \text{ implies } \vdash_{\Lambda} \varphi.$$

$\Lambda$  is strongly (weakly) complete with respect to a single structure  $\mathcal{S}$  if it is strongly (weakly) complete with respect to the class  $\mathbf{S} := \{\mathcal{S}\}$ .



## Normal logics - completeness

---

Obviously, weak completeness is a particular case of strong completeness; just take  $\Gamma = \emptyset$  in Definition 1.116.(i).

### Remark 1.117

$\Lambda$  is weakly complete with respect to  $\mathbf{S}$  iff  $\Lambda_{\mathbf{S}} \subseteq \Lambda$ .

If a normal logic  $\Lambda$  is both sound and weakly complete with respect to a class of structures  $\mathbf{S}$ , then there is a perfect match between the syntactic and semantic perspectives:  $\Lambda = \Lambda_{\mathbf{S}}$ .

Given a semantically specified normal logic  $\Lambda_{\mathbf{S}}$  (that is, the logic of some class of structures of interest), a very important problem is to find a simple set of formulas  $\Gamma$  such that  $\Lambda_{\mathbf{S}}$  is the logic generated by  $\Gamma$ ; we say that  $\Gamma$  **axiomatizes**  $\mathbf{S}$ .



## Completeness theorem for $K$

---

### *Theorem 1.118*

*$K$  is sound and strongly complete with respect to the class of all frames for  $ML_0$ .*



Let

$$(4) \quad \Box\varphi \rightarrow \Box\Box\varphi$$

We use the notation  $K4$  for the normal logic generated by (4).  
Thus,  $K4$  is the smallest normal logic that contains (4).

### *Theorem 1.119*

*$K4$  is sound and strongly complete with respect to the class of transitive frames.*



Let

$$(T) \quad \Box\varphi \rightarrow \varphi$$

We use the notation  $\mathcal{T}$  for the normal logic generated by  $(T)$ .

## *Definition 1.120*

We say that a frame  $\mathcal{F} = (W, R)$  is *reflexive* if  $R$  is reflexive.

## *Theorem 1.121*

$\mathcal{T}$  is sound and strongly complete with respect to the class of reflexive frames.



Let

$$(B) \quad \varphi \rightarrow \Box\Diamond\varphi$$

We use the notation  $B$  for the normal logic  $KB$  generated by  $(B)$ .

### *Definition 1.122*

We say that a frame  $\mathcal{F} = (W, R)$  is *symmetric* if  $R$  is symmetric.

### *Theorem 1.123*

$B$  is sound and strongly complete with respect to the class of symmetric frames.



Let

$$(D) \quad \Box\varphi \rightarrow \Diamond\varphi$$

$$(D') \quad \neg\Box(\varphi \wedge \neg\varphi)$$

One can easily see that  $\vdash_K (D) \leftrightarrow (D')$ .

Let  $\mathbf{KD}$  be the normal logic generated by  $(D)$  (or, equivalently, by  $(D')$ ).

### *Definition 1.124*

We say that a frame  $\mathcal{F} = (W, R)$  is **serial** if for all  $w \in W$  there exists  $v \in W$  such that  $Rwv$ .

### *Theorem 1.125*

$\mathbf{KD}$  is sound and strongly complete with respect to the class of serial frames.





Let

$$(5) \quad \Diamond\varphi \rightarrow \Box\Diamond\varphi$$

$$(5') \quad \neg\Box\varphi \rightarrow \Box\neg\Box\varphi$$

One can easily see that  $\vdash_K (5) \leftrightarrow (5')$ .

Let **K5** be the normal logic generated by (5) (or, equivalently, by (5')).

### Definition 1.126

We say that a frame  $\mathcal{F} = (W, R)$  is **Euclidean** if for all  $w, v, u \in W$ ,

if  $Rwv$  and  $Rwu$ , then  $Rvu$ .

### Theorem 1.127

**K5** is sound and strongly complete with respect to the class of Euclidean frames.



We use the notation  $S4$  for the normal logic  $KT4$  generated by  $(T)$  and  $(4)$ .

### *Theorem 1.128*

*$S4$  is sound and strongly complete with respect to the class of reflexive and transitive frames.*



We use the notation  $S5$  for the normal logic  $KT4B$  generated by  $(T)$ ,  $(4)$  and  $(B)$ .

### *Proposition 1.129*

$$S5 = KDB4 = KDB5 = KT5.$$

### *Theorem 1.130*

*$S5$  is sound and strongly complete with respect to the class of frames whose relation is an equivalence relation.*



The whole theory presented so far adapts easily to languages with more modal operators.

Let  $I$  be a nonempty set.

- ▶ The **multimodal language**  $ML_I$  consists of: a set  $PROP$  of atomic propositions,  $\neg$ ,  $\rightarrow$ ,  $\perp$ , the parentheses  $(, )$  and a set of modal operators  $\{\Box_i \mid i \in I\}$ .
- ▶ Formulas of  $ML_I$  are defined, using the Backus-Naur notation, as follows:

$$\varphi ::= p \mid \perp \mid (\neg\varphi) \mid (\varphi \rightarrow \psi) \mid (\Box_i\varphi),$$

where  $p \in PROP$  and  $i \in I$ .

- ▶ The dual of  $\Box_i$  is denoted by  $\Diamond_i$  and is defined as:

$$\Diamond_i\varphi := \neg\Box_i\neg\varphi$$



- ▶ A **frame** for  $ML_I$  is a relational structure  $\mathcal{F} = (W, \{R_i \mid i \in I\})$ , where  $R_i$  is a binary relation on  $W$  for every  $i \in I$ .
- ▶ A **model** for  $ML_I$  is, as previously, a pair  $\mathcal{M} = (\mathcal{F}, V)$ , where  $\mathcal{F}$  is a frame and  $V : PROP \rightarrow 2^W$  is a valuation.
- ▶ The last clause from the definition of the satisfaction relation  $\mathcal{M}, w \Vdash \varphi$  is changed to: for all  $i \in I$ ,  
 $\mathcal{M}, w \Vdash \Box_i \varphi$  iff for every  $v \in W$ ,  $R_i wv$  implies  $\mathcal{M}, v \Vdash \varphi$ .
- ▶ It follows that  
 $\mathcal{M}, w \Vdash \Diamond_i \varphi$  iff there exists  $v \in W$  s.t.  $R_i wv$  and  $\mathcal{M}, v \Vdash \varphi$ .
- ▶ The definitions of **truth in a model** ( $\mathcal{M} \Vdash \varphi$ ), of **validity in a frame** ( $\mathcal{F} \Vdash \varphi$ ) and of the **consequence relation** are unchanged.



### Definition 1.131

A **normal multimodal logic** is a set  $\Lambda$  of formulas of  $ML_I$  satisfying the following properties:

- ▶  $\Lambda$  contains all propositional tautologies and is closed under modus ponens.
- ▶  $\Lambda$  contains all formulas

$$(K_i) \quad \Box_i(\varphi \rightarrow \psi) \rightarrow (\Box_i\varphi \rightarrow \Box_i\psi),$$

where  $\varphi, \psi$  are formulas and  $i \in I$ .

- ▶  $\Lambda$  is closed under generalization: for any formula  $\varphi$  and all  $i \in I$ ,

$$\frac{\varphi}{\Box_i\varphi}.$$



- ▶ We use the same notation,  $\mathbf{K}$ , for the smallest normal multimodal logic.
- ▶ We define similarly  $\mathbf{K}$ -proofs and we also have that  $\mathbf{K} = \{\varphi \mid \vdash_{\mathbf{K}} \varphi\}$ .
- ▶ The multimodal logic generated by a set of formulas  $\Gamma$  is also denoted by  $\mathbf{K}\Gamma$ . Furthermore,  $\mathbf{K}\Gamma = \{\varphi \mid \vdash_{\mathbf{K}\Gamma} \varphi\}$ .
- ▶ The definitions of  $\Lambda$ -deducibility,  $\Lambda$ -consistence, soundness and (weak) completeness are unchanged.



# Epistemic Logics





In epistemic logics, the multimodal language is used to reason about knowledge. Let  $n \geq 1$  and  $AG = \{1, \dots, n\}$  be the set of agents.

- ▶ We consider the multimodal language  $ML_{Ag}$ .
- ▶ We write, for every  $i = 1, \dots, n$ ,  $K_i\varphi$  instead of  $\Box_i\varphi$ .
- ▶  $K_i\varphi$  is read as **the agent  $i$  knows (that)  $\varphi$** .
- ▶ We denote by  $\hat{K}_i$  the dual operator:  $\hat{K}_i\varphi = \neg K_i\neg\varphi$ .
- ▶ Then  $\hat{K}_i\varphi$  is read as **the agent  $i$  considers possible (that)  $\varphi$** .



### Definition 1.132

An **epistemic logic** is a set  $\Lambda$  of formulas of  $ML_{Ag}$  satisfying the following properties:

- ▶  $\Lambda$  contains all propositional tautologies and is closed under modus ponens.
- ▶  $\Lambda$  contains all formulas

$$K_i(\varphi \rightarrow \psi) \rightarrow (K_i\varphi \rightarrow K_i\psi),$$

where  $\varphi, \psi$  are formulas and  $i \in Ag$ .

- ▶  $\Lambda$  is closed under generalization: for any formula  $\varphi$  and all  $i \in Ag$ ,

$$\frac{\varphi}{K_i\varphi}.$$

We denote by **K** the smallest epistemic logic.



Recall the following axioms:

$$(T) \quad K_i \varphi \rightarrow \varphi$$

$$(D) \quad \neg K_i(\varphi \wedge \neg \varphi)$$

$$(B) \quad \varphi \rightarrow K_i \neg K_i \neg \varphi$$

$$(4) \quad K_i \varphi \rightarrow K_i K_i \varphi$$

$$(5) \quad \neg K_i \varphi \rightarrow K_i \neg K_i \varphi$$

### Properties of knowledge

- ▶ Axiom (T) is called the **verity** or **knowledge** axiom: If an agent knows  $\varphi$ , then  $\varphi$  must hold. **What is known is true.** This is often taken to be the property that distinguishes knowledge from other informational attitudes, such as belief.
- ▶ Axiom (D) is the **consistency** axiom: an agent does not know both  $\varphi$  and  $\neg \varphi$ . **An agent cannot know a contradiction.**



### Properties of knowledge

- ▶ Axiom (B) says that if  $\varphi$  holds, then an agent knows that it does not know  $\neg\varphi$ .
- ▶ Axiom (4) is **positive introspection**: if an agent knows  $\varphi$ , it knows that it knows  $\varphi$ . **An agent knows what it knows.**
- ▶ Axiom (5) is **negative introspection**: if an agent does not know  $\varphi$ , it knows that it does not know  $\varphi$ . **An agent is aware of what it doesn't know.**
- ▶ Positive and negative introspection together imply that an agent has perfect knowledge about what it does and does not know.



Let  $S5 = KDB4 = KDB5 = KT5$ .  $S5$  is considered as the logic of **idealised knowledge**.

### *Theorem 1.133*

*$S5$  is sound and strongly complete with respect to the class of frames whose relations are equivalence relations.*



## *Reasoning about knowledge*

---

- ▶ Consider a multiagent system, in which multiple agents autonomously perform some joint action.
- ▶ The agents need to communicate with one another.
- ▶ Problems appear when the communication is error-prone.
- ▶ One could have scenarios like the following:
  - ▶ Agent *A* sent the message to agent *B*.
  - ▶ The message may not arrive, and agent *A* knows this.
  - ▶ Furthermore, this is common knowledge, so agent *A* knows that agent *B* knows that *A* knows that if a message was sent it may not arrive.

### *Example 1.134*

Multiagent system = distributed system; agent = processor;  
action = computation

We use **epistemic logic** to make such reasoning precise.



## *Muddy children puzzle*

---

- ▶ A group of  $n$  children enters their house after having played in the mud outside. They are greeted in the hallway by their father, who notices that  $k$  of the children have mud on their foreheads.
- ▶ He makes the following announcement, “At least one of you has mud on his forehead.”
- ▶ The children can all see each other's foreheads, but not their own.
- ▶ The father then says, “Do any of you know that you have mud on your forehead? If you do, raise your hand now.”
- ▶ No one raises his hand.
- ▶ The father repeats the question, and again no one moves.
- ▶ The father does not give up and keeps repeating the question.
- ▶ After exactly  $k$  repetitions, all the children with muddy foreheads raise their hands simultaneously.



## *Muddy children puzzle*

---

$k = 1$

- ▶ There exists only one muddy child.
- ▶ The muddy child knows the other children are clean.
- ▶ When the father says at least one is muddy, he concludes that it's him.
- ▶ None of the other children know at this point whether or not they are muddy.
- ▶ The muddy child raises his hand after the father's first question.
- ▶ After the muddy child raises his hand, the other children know that they are clean.





## *Muddy children puzzle*

---

$$k = 2$$

- ▶ There exist two muddy children.
- ▶ Imagine that you are one of the two muddy children.
- ▶ You see that one of the other children is muddy.
- ▶ After the father's first announcement, you do not have enough information to know whether you are muddy. You might be, but it could also be that the other child is the only muddy one.
- ▶ So, you do not raise the hand after the father's first question.
- ▶ You note that the other muddy child does not raise his hand.
- ▶ You realize then that you yourself must be muddy as well, or else that child would have raised his hand.
- ▶ So, after the father's second question, you raise your hand. Of course, so does the other muddy child.



## *Muddy children puzzle*

---

- ▶ One could extend this argument to  $k = 3, 4, \dots$
- ▶ Of course, one would rather have a general theorem that applies to all  $k$ .
- ▶ For this we will need a **formal model of “know”** that applies in this example.



Partition models of knowledge are defined by Shoham and Leyton-Brown in [3]. Let  $n \geq 1$  and  $AG = \{1, \dots, n\}$  be the set of agents.

### *Definition 1.135 (Partition frame)*

A *partition frame* is a tuple  $\mathcal{P}_F = (W, I_1, \dots, I_n)$ , where

- ▶  $W$  is a nonempty set of *possible worlds*.
- ▶ For every  $i = 1, \dots, n$ ,  $I_i$  is a *partition* of  $W$ .

The idea is that  $I_i$  partitions  $W$  into sets of possible worlds that are *indistinguishable* from the point of view of agent  $i$ .



## Partition model of knowledge

---

**Recall:** Let  $A$  be a nonempty set. A **partition** of  $A$  is a family  $(A_j)_{j \in J}$  of nonempty subsets of  $A$  satisfying the following properties:

$$A = \bigcup_{j \in J} A_j \text{ and } A_j \cap A_k = \emptyset \text{ for all } j \neq k.$$

**Recall:** Let  $A$  be a nonempty set. There exists a bijection between the set of partitions of  $A$  and the set of equivalence relations on  $A$ :

- ▶  $(A_j)_{j \in J}$  partition of  $A \mapsto$  the equivalence relation on  $A$  defined by  $x \sim y \Leftrightarrow$  there exists  $j \in J$  such that  $x, y \in A_j$ .
- ▶  $\sim$  equivalence relation on  $A \mapsto$  the partition consisting of all the different equivalence classes of  $\sim$ .



## *Partition model of knowledge*

---

- ▶ For each  $i = 1, \dots, n$ , let  $R_{I_i}$  be the corresponding equivalence relation.
- ▶ Denote by  $I_i(w)$  the equivalence class of  $w$  in the relation  $R_{I_i}$ .
- ▶ If the actual world is  $w$ , then  $I_i(w)$  is the set of possible worlds that agent  $i$  cannot distinguish from  $w$ .
- ▶  $\mathcal{F} = (W, R_{I_1}, \dots, R_{I_n})$  is a frame for the epistemic logic **S5**.

Partition frame = frame for the epistemic logic **S5**



### Definition 1.136 (Partition model)

A **partition model** over a language  $\Sigma$  is a tuple  $\mathcal{P}_M = (\mathcal{P}_F, \pi)$ , where

- ▶  $\mathcal{P}_F = (W, I_1, \dots, I_n)$  is a partition frame.
- ▶  $\pi : \Sigma \rightarrow 2^W$  is an interpretation function.

For every statement  $\varphi \in \Sigma$ , we think of  $\pi(\varphi)$  as the set of possible worlds in the partition model  $\mathcal{P}_M$  where  $\varphi$  is satisfied.

- ▶ Each possible world completely specifies the concrete state of affairs.
- ▶ We can take, for example,  $\Sigma$  to be a set of formulas in propositional logic over some set of atomic propositions.



## Partition model of knowledge

We will use the notation  $K_i\varphi$  as “agent  $i$  knows that  $\varphi$ ”.

The following defines when a statement is true in a partition model.

### *Definition 1.137 (Logical entailment for partition models)*

Let  $\mathcal{P}_M = (W, I_1, \dots, I_n, \pi)$  be a partition model over  $\Sigma$ , and  $w \in W$ . We define the  $\models$  (logical entailment) relation as follows:

- ▶ For any  $\varphi \in \Sigma$ , we say that  $\mathcal{P}_M, w \models \varphi$  iff  $w \in \pi(\varphi)$ .
- ▶  $\mathcal{P}_M, w \models K_i\varphi$  iff for all worlds  $v \in W$ , if  $v \in I_i(w)$ , then  $\mathcal{P}_M, v \models \varphi$ .

Partition model = model for the epistemic logic **S5**

We can reason about knowledge rigorously in terms of partition models, hence using epistemic logic.



## Partition model of knowledge

---

We apply the partition model of knowledge to the Muddy Children puzzle.

- ▶ We consider the case  $n = k = 2$  (two children, both muddy).
- ▶ There are two atomic propositions: **muddy1** and **muddy2**.
- ▶ There are four possible worlds, corresponding to each of the children being muddy or not:
  - $w_1 : \text{muddy1} \wedge \text{muddy2}$  (real world)
  - $w_2 : \text{muddy1} \wedge \neg \text{muddy2}$
  - $w_3 : \neg \text{muddy1} \wedge \text{muddy2}$
  - $w_4 : \neg \text{muddy1} \wedge \neg \text{muddy2}$ .
- ▶ Thus,  $\pi(\text{muddy1}) = \{w_1, w_2\}$  and  $\pi(\text{muddy2}) = \{w_1, w_3\}$ .
- ▶ There are two partitions  $I_1$  and  $I_2$ .





## Partition model of knowledge

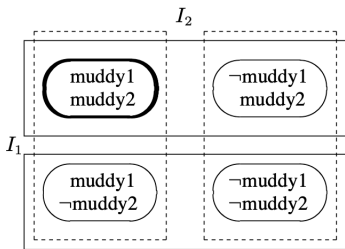


Figure 7: Partition model after the children see each other

- ▶ The ovals illustrate the four possible worlds, with the dark oval indicating the true state of the world.
- ▶ The solid boxes indicate the equivalence classes in  $I_1$ :  
 $I_1(w_1) = I_1(w_3) = \{w_1, w_3\}$ ,  $I_1(w_2) = I_1(w_4) = \{w_2, w_4\}$
- ▶ The dashed boxes indicate the equivalence classes in  $I_2$ :  
 $I_2(w_1) = I_2(w_2) = \{w_1, w_2\}$ ,  $I_2(w_3) = I_2(w_4) = \{w_3, w_4\}$ .
- ▶ In the real world  $w_1$ ,  $K_1$ muddy2 and  $K_2$ muddy1 are true; neither  $K_1$ muddy1 nor  $K_2$ muddy2 is true.



## Partition model of knowledge

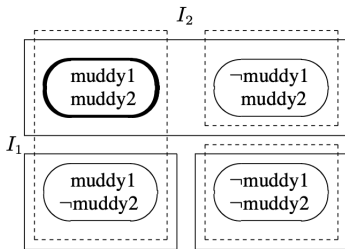


Figure 8: Partition model after the father's announcement

- ▶ The world in which neither child is muddy is ruled out. The state of knowledge is as shown in Figure 8.  
$$I_1(w_1) = I_1(w_3) = \{w_1, w_3\}, \quad I_1(w_2) = \{w_2\}, \quad I_1(w_4) = \{w_4\}$$
$$I_2(w_1) = I_2(w_2) = \{w_1, w_2\}, \quad I_2(w_3) = \{w_3\}, \quad I_2(w_4) = \{w_4\}.$$
- ▶ In the real world  $w_1$ , it is still the case that neither  $K_1 \text{muddy1}$  nor  $K_2 \text{muddy2}$  is true.



## Partition model of knowledge

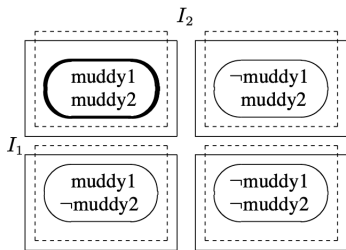


Figure 9: Final partition model

- ▶ Once the children each observe that the other child does not raise his hand after the father's question, the state of knowledge becomes as shown in Figure 9:
- ▶  $I_k(w_i) = \{w_i\}$  for all  $k = 1, 2, i = 1, \dots, 4$ .
- ▶ Both  $K_1 \text{muddy1}$  and  $K_2 \text{muddy2}$  hold now in the real world  $w_1$ .