

Logic for Multiagent Systems

Master 1st Year, 1st Semester 2023/2024

Laurențiu Leuștean

Web page: <https://cs.unibuc.ro/~lleustean/Teaching/2023-LogMA/index.html>

1

Propositional logic

2

Language

Definition 1.1

The language of *propositional logic PL* consists of:

- ▶ a countable set $V = \{v_n \mid n \in \mathbb{N}\}$ of variables;
- ▶ the logic connectives \neg (*non*), \rightarrow (*implies*)
- ▶ parantheses: $(,)$.

- The set *Sym* of *symbols* of PL is

$$\text{Sym} := V \cup \{\neg, \rightarrow, (,)\}.$$

- We denote variables by $u, v, x, y, z \dots$

3

Language

Definition 1.2

The set *Expr* of *expressions* of PL is the set of all finite sequences of symbols of PL.

Definition 1.3

Let $\theta = \theta_0\theta_1 \dots \theta_{k-1}$ be an expression, where $\theta_i \in \text{Sym}$ for all $i = 0, \dots, k-1$.

- ▶ If $0 \leq i \leq j \leq k-1$, then the expression $\theta_i \dots \theta_j$ is called the *(i, j)-subexpression* of θ .
- ▶ We say that an expression ψ *appears* in θ if there exists $0 \leq i \leq j \leq k-1$ such that ψ is the *(i, j)-subexpression* of θ .
- ▶ We denote by $\text{Var}(\theta)$ the set of variables appearing in θ .

4

The definition of formulas is an example of an **inductive definition**.

Definition 1.4

The **formulas** of PL are the expressions of PL defined as follows:

- (F0) Any variable is a formula.
- (F1) If φ is a formula, then $(\neg\varphi)$ is a formula.
- (F2) If φ and ψ are formulas, then $(\varphi \rightarrow \psi)$ is a formula.
- (F3) Only the expressions obtained by applying rules (F0), (F1), (F2) are formulas.

Notations

The set of formulas is denoted by **Form**. Formulas are denoted by $\varphi, \psi, \chi, \dots$

Proposition 1.5

The set **Form** is countable.

Unique readability

If φ is a formula, then **exactly** one of the following hold:

- ▶ $\varphi = v$, where $v \in V$.
- ▶ $\varphi = (\neg\psi)$, where ψ is a formula.
- ▶ $\varphi = (\psi \rightarrow \chi)$, where ψ, χ are formulas.

Furthermore, φ can be written in a unique way in one of these forms.

Definition 1.6

Let φ be a formula. A **subformula** of φ is any formula ψ that appears in φ .

Proposition 1.7 (Induction principle on formulas)

Let Γ be a set of formulas satisfying the following properties:

- ▶ $V \subseteq \Gamma$.
- ▶ Γ is closed to \neg , that is: $\varphi \in \Gamma$ implies $(\neg\varphi) \in \Gamma$.
- ▶ Γ is closed to \rightarrow , that is: $\varphi, \psi \in \Gamma$ implies $(\varphi \rightarrow \psi) \in \Gamma$.

Then $\Gamma = \text{Form}$.

It is used to prove that all formulas have a property \mathcal{P} : we define Γ as the set of all formulas satisfying \mathcal{P} and apply induction on formulas to obtain that $\Gamma = \text{Form}$.

The derived connectives \vee (**or**), \wedge (**and**), \leftrightarrow (**if and only if**) are introduced by the following abbreviations:

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

Conventions and notations

- ▶ The external parantheses are omitted, we put them only when necessary. We write $\neg\varphi$, $\varphi \rightarrow \psi$, but we write $(\varphi \rightarrow \psi) \rightarrow \chi$.
- ▶ To reduce the use of parantheses, we assume that
 - ▶ \neg has higher precedence than $\rightarrow, \wedge, \vee, \leftrightarrow$;
 - ▶ \wedge, \vee have higher precedence than $\rightarrow, \leftrightarrow$.
- ▶ Hence, the formula $((\varphi \rightarrow (\psi \vee \chi)) \wedge ((\neg\psi) \leftrightarrow (\psi \vee \chi)))$ is written as $(\varphi \rightarrow \psi \vee \chi) \wedge (\neg\psi \leftrightarrow \psi \vee \chi)$.

Truth values

We use the following notations for the truth values:

1 for **true** and 0 for **false**.

Hence, the set of truth values is $\{0, 1\}$.

Define the following operations on $\{0, 1\}$ using **truth tables**.

$$\neg : \{0, 1\} \rightarrow \{0, 1\},$$

p	$\neg p$
0	1
1	0

$$\rightarrow : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\},$$

p	q	$p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

9

$$\vee : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\},$$

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

$$\wedge : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\},$$

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

$$\leftrightarrow : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\},$$

p	q	$p \leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

10

Definition 1.8

An **evaluation** (or **interpretation**) is a function $e : V \rightarrow \{0, 1\}$.

Theorem 1.9

For any evaluation $e : V \rightarrow \{0, 1\}$ there exists a unique function

$$e^+ : \text{Form} \rightarrow \{0, 1\}$$

satisfying the following properties:

- ▶ $e^+(v) = e(v)$ for all $v \in V$.
- ▶ $e^+(\neg\varphi) = \neg e^+(\varphi)$ for any formula φ .
- ▶ $e^+(\varphi \rightarrow \psi) = e^+(\varphi) \rightarrow e^+(\psi)$ for any formulas φ, ψ .

Proposition 1.10

For any formula φ and all evaluations $e_1, e_2 : V \rightarrow \{0, 1\}$,
if $e_1(v) = e_2(v)$ for all $v \in \text{Var}(\varphi)$, then $e_1^+(\varphi) = e_2^+(\varphi)$.

11

Let φ be a formula.

Definition 1.11

- ▶ An evaluation $e : V \rightarrow \{0, 1\}$ is a **model** of φ if $e^+(\varphi) = 1$.
Notation: $e \models \varphi$.
- ▶ φ is **satisfiable** if it has a model.
- ▶ If φ is not satisfiable, we also say that φ is **unsatisfiable** or **contradictory**.
- ▶ φ is a **tautology** if every evaluation is a model of φ .
Notation: $\models \varphi$.

Notation 1.12

The set of models of φ is denoted by $\text{Mod}(\varphi)$.

12

Remark 1.13

- ▶ φ is a tautology iff $\neg\varphi$ is unsatisfiable.
- ▶ φ is unsatisfiable iff $\neg\varphi$ is a tautology.

Proposition 1.14

Let $e : V \rightarrow \{0, 1\}$ be an evaluation. Then for all formulas φ, ψ ,

- ▶ $e \models \neg\varphi$ iff $e \not\models \varphi$.
- ▶ $e \models \varphi \rightarrow \psi$ iff ($e \models \varphi$ implies $e \models \psi$) iff ($e \not\models \varphi$ or $e \models \psi$).
- ▶ $e \models \varphi \vee \psi$ iff ($e \models \varphi$ or $e \models \psi$).
- ▶ $e \models \varphi \wedge \psi$ iff ($e \models \varphi$ and $e \models \psi$).
- ▶ $e \models \varphi \leftrightarrow \psi$ iff ($e \models \varphi$ iff $e \models \psi$).

13

Definition 1.15

Let φ, ψ be formulas. We say that

- ▶ φ is a **semantic consequence** of ψ if $\text{Mod}(\psi) \subseteq \text{Mod}(\varphi)$.
Notation: $\psi \models \varphi$.
- ▶ φ and ψ are **(logically) equivalent** if $\text{Mod}(\psi) = \text{Mod}(\varphi)$.
Notation: $\varphi \sim \psi$.

Remark 1.16

Let φ, ψ be formulas.

- ▶ $\psi \models \varphi$ iff $\models \psi \rightarrow \varphi$.
- ▶ $\psi \sim \varphi$ iff ($\psi \models \varphi$ and $\varphi \models \psi$) iff $\models \psi \leftrightarrow \varphi$.

14

For all formulas φ, ψ, χ ,

- $\models \varphi \vee \neg\varphi$
- $\models \neg(\varphi \wedge \neg\varphi)$
- $\models \varphi \wedge \psi \rightarrow \varphi$
- $\models \varphi \rightarrow \varphi \vee \psi$
- $\models \varphi \rightarrow (\psi \rightarrow \varphi)$
- $\models (\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$
- $\models (\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \chi))$
- $\models (\varphi \rightarrow \psi) \vee (\neg\varphi \rightarrow \psi)$
- $\models (\varphi \rightarrow \psi) \vee (\varphi \rightarrow \neg\psi)$
- $\models \neg\varphi \rightarrow (\neg\psi \leftrightarrow (\psi \rightarrow \varphi))$
- $\models (\varphi \rightarrow \psi) \rightarrow (((\varphi \rightarrow \chi) \rightarrow \psi) \rightarrow \psi)$
- $\models \neg\psi \rightarrow (\psi \rightarrow \varphi)$

15

- $\models \psi \rightarrow (\neg\psi \rightarrow \varphi)$
- $\models (\varphi \rightarrow \neg\varphi) \rightarrow \neg\varphi$
- $\models (\neg\varphi \rightarrow \varphi) \rightarrow \varphi$
- $\psi \models \varphi \rightarrow \psi$
- $\neg\varphi \models \varphi \rightarrow \psi$
- $\neg\psi \wedge (\varphi \rightarrow \psi) \models \neg\varphi$
- $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \chi) \models \varphi \rightarrow \chi$
- $\varphi \wedge (\varphi \rightarrow \psi) \models \psi$

16

$$\begin{aligned}
\varphi \vee \psi &\sim \neg(\neg\varphi \wedge \neg\psi) \\
\varphi \wedge \psi &\sim \neg(\neg\varphi \vee \neg\psi) \\
\varphi \rightarrow (\psi \rightarrow \chi) &\sim \varphi \wedge \psi \rightarrow \chi \\
\varphi &\sim \varphi \wedge \varphi \sim \varphi \vee \varphi \\
\varphi \wedge \psi &\sim \psi \wedge \varphi \\
\varphi \vee \psi &\sim \psi \vee \varphi \\
\varphi \wedge (\psi \wedge \chi) &\sim (\varphi \wedge \psi) \wedge \chi \\
\varphi \vee (\psi \vee \chi) &\sim (\varphi \vee \psi) \vee \chi \\
\varphi \vee (\varphi \wedge \psi) &\sim \varphi \\
\varphi \wedge (\varphi \vee \psi) &\sim \varphi
\end{aligned}$$

17

$$\begin{aligned}
\varphi \wedge (\psi \vee \chi) &\sim (\varphi \wedge \psi) \vee (\varphi \wedge \chi) \\
\varphi \vee (\psi \wedge \chi) &\sim (\varphi \vee \psi) \wedge (\varphi \vee \chi) \\
\varphi \rightarrow \psi \wedge \chi &\sim (\varphi \rightarrow \psi) \wedge (\varphi \rightarrow \chi) \\
\varphi \rightarrow \psi \vee \chi &\sim (\varphi \rightarrow \psi) \vee (\varphi \rightarrow \chi) \\
\varphi \wedge \psi \rightarrow \chi &\sim (\varphi \rightarrow \chi) \vee (\psi \rightarrow \chi) \\
\varphi \vee \psi \rightarrow \chi &\sim (\varphi \rightarrow \chi) \wedge (\psi \rightarrow \chi) \\
\varphi \rightarrow (\psi \rightarrow \chi) &\sim \psi \rightarrow (\varphi \rightarrow \chi) \\
&\sim (\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi) \\
\neg\varphi \sim \varphi \rightarrow \neg\varphi &\sim (\varphi \rightarrow \psi) \wedge (\varphi \rightarrow \neg\psi) \\
\varphi \rightarrow \psi \sim \neg\varphi \vee \psi &\sim \neg(\varphi \wedge \neg\psi) \\
\varphi \vee \psi \sim \varphi \vee (\neg\varphi \wedge \psi) &\sim (\varphi \rightarrow \psi) \rightarrow \psi \\
\varphi \leftrightarrow (\psi \leftrightarrow \chi) &\sim (\varphi \leftrightarrow \psi) \leftrightarrow \chi
\end{aligned}$$

18

It is often useful to have a canonical tautology and a canonical unsatisfiable formula.

Remark 1.17

$v_0 \rightarrow v_0$ is a tautology and $\neg(v_0 \rightarrow v_0)$ is unsatisfiable.

Notation 1.18

Denote $v_0 \rightarrow v_0$ by \top and call it *the truth*.

Denote $\neg(v_0 \rightarrow v_0)$ by \perp and call it *the false*.

Remark 1.19

- ▶ φ is a tautology iff $\varphi \sim \top$.
- ▶ φ is unsatisfiable iff $\varphi \sim \perp$.

19

Let Γ be a set of formulas.

Definition 1.20

An evaluation $e : V \rightarrow \{0, 1\}$ is a *model* of Γ if it is a model of every formula from Γ .

Notation: $e \models \Gamma$.

Notation 1.21

The set of models of Γ is denoted by $\text{Mod}(\Gamma)$.

Definition 1.22

A formula φ is a *semantic consequence* of Γ if $\text{Mod}(\Gamma) \subseteq \text{Mod}(\varphi)$.

Notation: $\Gamma \models \varphi$.

20

Definition 1.23

- ▶ Γ is **satisfiable** if it has a model.
- ▶ Γ is **finitely satisfiable** if every finite subset of Γ is satisfiable.
- ▶ If Γ is not satisfiable, we say also that Γ is **unsatisfiable** or **contradictory**.

Proposition 1.24

The following are equivalent:

- ▶ Γ is unsatisfiable.
- ▶ $\Gamma \models \perp$.

Theorem 1.25 (Compactness Theorem)

Γ is satisfiable iff Γ is finitely satisfiable.

21

We use a **deductive system** of Hilbert type for LP .

Logical axioms

The set Axm of **(logical) axioms** of LP consists of:

- (A1) $\varphi \rightarrow (\psi \rightarrow \varphi)$
- (A2) $(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$
- (A3) $(\neg\psi \rightarrow \neg\varphi) \rightarrow (\varphi \rightarrow \psi)$,

where φ , ψ and χ are formulas.

The deduction rule

For any formulas φ , ψ , from φ and $\varphi \rightarrow \psi$ infer ψ (**modus ponens** or **(MP)**):

$$\frac{\varphi, \varphi \rightarrow \psi}{\psi}$$

22

Let Γ be a set of formulas. The definition of Γ -theorems is another example of an inductive definition.

Definition 1.26

The **Γ -theorems** of PL are the formulas defined as follows:

- (T0) Every logical axiom is a Γ -theorem.
- (T1) Every formula of Γ is a Γ -theorem.
- (T2) If φ and $\varphi \rightarrow \psi$ are Γ -theorems, then ψ is a Γ -theorem.
- (T3) Only the formulas obtained by applying rules (T0), (T1), (T2) are Γ -theorems.

If φ is a Γ -theorem, then we also say that φ is **deduced from the hypotheses** Γ .

23

Notations

- $\Gamma \vdash \varphi$: \Leftrightarrow φ is a Γ -theorem
- $\vdash \varphi$: \Leftrightarrow $\emptyset \vdash \varphi$.

Definition 1.27

A formula φ is called a **theorem** of LP if $\vdash \varphi$.

By a reformulation of the conditions (T0), (T1), (T2) using the notation \vdash , we get

Remark 1.28

- ▶ If φ is an axiom, then $\Gamma \vdash \varphi$.
- ▶ If $\varphi \in \Gamma$, then $\Gamma \vdash \varphi$.
- ▶ If $\Gamma \vdash \varphi$ and $\Gamma \vdash \varphi \rightarrow \psi$, then $\Gamma \vdash \psi$.

24

Definition 1.29

A Γ -proof (or *proof from the hypotheses Γ*) is a sequence of formulas $\theta_1, \dots, \theta_n$ such that for all $i \in \{1, \dots, n\}$, one of the following holds:

- ▶ θ_i is an axiom.
- ▶ $\theta_i \in \Gamma$.
- ▶ there exist $k, j < i$ such that $\theta_k = \theta_j \rightarrow \theta_i$.

Definition 1.30

Let φ be a formula. A Γ -proof of φ or a *proof of φ from the hypotheses Γ* is a Γ -proof $\theta_1, \dots, \theta_n$ such that $\theta_n = \varphi$.

Proposition 1.31

For any formula φ ,

$\Gamma \vdash \varphi$ iff there exists a Γ -proof of φ .

25

Theorem 1.32 (Deduction Theorem)

Let $\Gamma \cup \{\varphi, \psi\}$ be a set of formulas. Then

$\Gamma \cup \{\varphi\} \vdash \psi$ iff $\Gamma \vdash \varphi \rightarrow \psi$.

Proposition 1.33

For any formulas φ, ψ, χ ,

$\vdash (\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \chi))$

$\vdash (\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow (\psi \rightarrow (\varphi \rightarrow \chi))$

Proposition 1.34

Let $\Gamma \cup \{\varphi, \psi, \chi\}$ be a set of formulas.

$\Gamma \vdash \varphi \rightarrow \psi$ and $\Gamma \vdash \psi \rightarrow \chi \Rightarrow \Gamma \vdash \varphi \rightarrow \chi$

$\Gamma \cup \{\neg\psi\} \vdash \neg(\varphi \rightarrow \varphi) \Rightarrow \Gamma \vdash \psi$

$\Gamma \cup \{\psi\} \vdash \varphi$ and $\Gamma \cup \{\neg\psi\} \vdash \varphi \Rightarrow \Gamma \vdash \varphi$.

26

Let Γ be a set of formulas.

Definition 1.35

Γ is called *consistent* if there exists a formula φ such that $\Gamma \not\vdash \varphi$.

Γ is said to be *inconsistent* if it is not consistent, that is $\Gamma \vdash \varphi$ for any formula φ .

Proposition 1.36

- ▶ \emptyset is consistent.
- ▶ The set of theorems is consistent.

Proposition 1.37

The following are equivalent:

- ▶ Γ is inconsistent.
- ▶ $\Gamma \vdash \perp$.

27

Theorem 1.38 (Completeness Theorem (version 1))

Let Γ be a set of formulas. Then

Γ is consistent \iff Γ is satisfiable.

Theorem 1.39 (Completeness Theorem (version 2))

Let Γ be a set of formulas. For any formula φ ,

$\Gamma \vdash \varphi \iff \Gamma \models \varphi$.

28

First-order logic

29

First-order languages

Definition 2.1

A *first-order language* \mathcal{L} consists of:

- ▶ a countable set $V = \{v_n \mid n \in \mathbb{N}\}$ of variables;
 - ▶ the connectives \neg and \rightarrow ;
 - ▶ parentheses $(,)$;
 - ▶ the equality symbol $=$;
 - ▶ the universal quantifier \forall ;
 - ▶ a set \mathcal{R} of *relation symbols*;
 - ▶ a set \mathcal{F} of *function symbols*;
 - ▶ a set \mathcal{C} of *constant symbols*;
 - ▶ an *arity* function $\text{ari} : \mathcal{F} \cup \mathcal{R} \rightarrow \mathbb{N}^*$.
- ▶ \mathcal{L} is uniquely determined by the quadruple $\tau := (\mathcal{R}, \mathcal{F}, \mathcal{C}, \text{ari})$.
- ▶ τ is called the *signature* of \mathcal{L} or the *similaritay type* of \mathcal{L} .

30

First-order languages

Let \mathcal{L} be a first-order language.

- The set $\text{Sym}_{\mathcal{L}}$ of *symbols* of \mathcal{L} is

$$\text{Sym}_{\mathcal{L}} := V \cup \{\neg, \rightarrow, (,), =, \forall\} \cup \mathcal{R} \cup \mathcal{F} \cup \mathcal{C}$$

- The elements of $\mathcal{R} \cup \mathcal{F} \cup \mathcal{C}$ are called *non-logical symbols*.
- The elements of $V \cup \{\neg, \rightarrow, (,), =, \forall\}$ are called *logical symbols*.

- We denote variables by x, y, z, v, \dots , relation symbols by P, Q, R, \dots , function symbols by f, g, h, \dots and constant symbols by c, d, e, \dots

- For every $m \in \mathbb{N}^*$ we denote:

\mathcal{F}_m := the set of function symbols of arity m ;

\mathcal{R}_m := the set of relation symbols of arity m .

31

First-order languages

Definition 2.2

The set $\text{Expr}_{\mathcal{L}}$ of *expressions* of \mathcal{L} is the set of all finite sequences of symbols of \mathcal{L} .

Definition 2.3

Let $\theta = \theta_0\theta_1 \dots \theta_{k-1}$ be an expression of \mathcal{L} , where $\theta_i \in \text{Sym}_{\mathcal{L}}$ for all $i = 0, \dots, k-1$.

- ▶ If $0 \leq i \leq j \leq k-1$, then the expression $\theta_i \dots \theta_j$ is called the *(i, j) -subexpression* of θ .
- ▶ We say that an expression ψ *appears* in θ if there exists $0 \leq i \leq j \leq k-1$ such that ψ is the (i, j) -subexpression of θ .
- ▶ We denote by $\text{Var}(\theta)$ the set of variables appearing in θ .

32

Definition 2.4

The **terms** of \mathcal{L} are the expressions defined as follows:

- (T0) Every variable is a term.
- (T1) Every constant symbol is a term.
- (T2) If $m \geq 1$, $f \in \mathcal{F}_m$ and t_1, \dots, t_m are terms, then $ft_1 \dots t_m$ is a term.
- (T3) Only the expressions obtained by applying rules (T0), (T1), (T2) are terms.

Notations:

- ▶ The set of terms is denoted by $Term_{\mathcal{L}}$.
- ▶ Terms are denoted by $t, s, t_1, t_2, s_1, s_2, \dots$
- ▶ $Var(t)$ is the set of variables that appear in the term t .

Definition 2.5

A term t is called **closed** if $Var(t) = \emptyset$.

Proposition 2.6 (Induction on terms)

Let Γ be a set of terms satisfying the following properties:

- ▶ Γ contains the variables and the constant symbols.
- ▶ If $m \geq 1$, $f \in \mathcal{F}_m$ and $t_1, \dots, t_m \in \Gamma$, then $ft_1 \dots t_m \in \Gamma$.

Then $\Gamma = Term_{\mathcal{L}}$.

It is used to prove that all terms have a property \mathcal{P} : we define Γ as the set of all terms satisfying \mathcal{P} and apply induction on terms to obtain that $\Gamma = Term_{\mathcal{L}}$.

Definition 2.7

The **atomic formulas** of \mathcal{L} are the expressions having one of the following forms:

- ▶ $(s = t)$, where s, t are terms;
- ▶ $(Rt_1 \dots t_m)$, where $R \in \mathcal{R}_m$ and t_1, \dots, t_m are terms.

Definition 2.8

The **formulas** of \mathcal{L} are the expressions defined as follows:

- (F0) Every atomic formula is a formula.
- (F1) If φ is a formula, then $(\neg\varphi)$ is a formula.
- (F2) If φ and ψ are formulas, then $(\varphi \rightarrow \psi)$ is a formula.
- (F3) If φ is a formula, then $(\forall x\varphi)$ is a formula for every variable x .
- (F4) Only the expressions obtained by applying rules (F0), (F1), (F2), (F3) are formulas.

Notations

- ▶ The set of formulas is denoted by $Form_{\mathcal{L}}$.
- ▶ Formulas are denoted by $\varphi, \psi, \chi, \dots$
- ▶ $Var(\varphi)$ is the set of variables that appear in the formula φ .

Unique readability

If φ is a formula, then **exactly** one of the following hold:

- ▶ $\varphi = (s = t)$, where s, t are terms.
- ▶ $\varphi = (Rt_1 \dots t_m)$, where $R \in \mathcal{R}_m$ and t_1, \dots, t_m are terms.
- ▶ $\varphi = (\neg\psi)$, where ψ is a formula.
- ▶ $\varphi = (\psi \rightarrow \chi)$, where ψ, χ are formulas.
- ▶ $\varphi = (\forall x\psi)$, where x is a variable and ψ is a formula.

Furthermore, φ can be written in a unique way in one of these forms.

Proposition 2.9 (Induction principle on formulas)

Let Γ be a set of formulas satisfying the following properties:

- ▶ Γ contains all atomic formulas.
- ▶ Γ is closed to \neg, \rightarrow and $\forall x$ (for any variable x), that is:
if $\varphi, \psi \in \Gamma$, then $(\neg\varphi), (\varphi \rightarrow \psi), (\forall x\varphi) \in \Gamma$.

Then $\Gamma = \text{Form}_{\mathcal{L}}$.

It is used to prove that all formulas have a property \mathcal{P} : we define Γ as the set of all formulas satisfying \mathcal{P} and apply induction on formulas to obtain that $\Gamma = \text{Form}_{\mathcal{L}}$.

Derived connectives

Connectives $\vee, \wedge, \leftrightarrow$ and the **existential quantifier** \exists are introduced by the following abbreviations:

$$\begin{aligned} \varphi \vee \psi &:= ((\neg\varphi) \rightarrow \psi) \\ \varphi \wedge \psi &:= \neg(\varphi \rightarrow (\neg\psi)) \\ \varphi \leftrightarrow \psi &:= ((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)) \\ \exists x\varphi &:= (\neg\forall x(\neg\varphi)) \end{aligned}$$

Usually the external parantheses are omitted, we write them only when necessary. We write $s = t, Rt_1 \dots t_m, ft_1 \dots t_m, \neg\varphi, \varphi \rightarrow \psi, \forall x\varphi$. On the other hand, we write $(\varphi \rightarrow \psi) \rightarrow \chi$.

To reduce the use of parentheses, we assume that

- ▶ \neg has higher precedence than $\rightarrow, \wedge, \vee, \leftrightarrow$;
- ▶ \wedge, \vee have higher precedence than $\rightarrow, \leftrightarrow$;
- ▶ quantifiers \forall, \exists have higher precedence than the other connectives. Thus, $\forall x\varphi \rightarrow \psi$ is $(\forall x\varphi) \rightarrow \psi$ and not $\forall x(\varphi \rightarrow \psi)$.

- ▶ We write sometimes $f(t_1, \dots, t_m)$ instead of $ft_1 \dots t_m$ and $R(t_1, \dots, t_m)$ instead of $Rt_1 \dots t_m$.
- ▶ Function/relation symbols of arity 1 are called **unary**. Function/relation symbols of arity 2 are called **binary**.
- ▶ If f is a binary function symbol, we write t_1ft_2 instead of ft_1t_2 .
- ▶ If R is a binary relation symbol, we write t_1Rt_2 instead of Rt_1t_2 .

We identify often a language \mathcal{L} with the set of its non-logical symbols and write $\mathcal{L} = (\mathcal{R}, \mathcal{F}, \mathcal{C})$.

Definition 2.10

Let $\varphi = \varphi_0\varphi_1 \dots \varphi_{n-1}$ be a formula of \mathcal{L} and x be a variable.

- ▶ We say that x **occurs bound on position k** in φ if $x = \varphi_k$ and there exists $0 \leq i \leq k \leq j \leq n - 1$ such that the (i, j) -subexpression of φ has the form $\forall x\psi$.
- ▶ We say that x **occurs free on position k** in φ if $x = \varphi_k$, but x does not occur bound on position k in φ .
- ▶ x is a **bound variable** of φ if there exists k such that x occurs bound on position k in φ .
- ▶ x is a **free variable** of φ if there exists k such that x occurs free on position k in φ .

Example

Let $\varphi = \forall x(x = y) \rightarrow x = z$. Free variables: x, y, z . Bound variables: x .

41

Notation: $FV(\varphi) :=$ the set of free variables of φ .

Alternative definition

The set $FV(\varphi)$ of free variables of a formula φ can be also defined by induction on formulas:

$$\begin{aligned} FV(\varphi) &= \text{Var}(\varphi), & \text{if } \varphi \text{ is an atomic formula} \\ FV(\neg\varphi) &= FV(\varphi) \\ FV(\varphi \rightarrow \psi) &= FV(\varphi) \cup FV(\psi) \\ FV(\forall x\varphi) &= FV(\varphi) \setminus \{x\}. \end{aligned}$$

42

Definition 2.11

An \mathcal{L} -**structure** is a quadruple

$$\mathcal{A} = (A, \mathcal{F}^{\mathcal{A}}, \mathcal{R}^{\mathcal{A}}, \mathcal{C}^{\mathcal{A}}),$$

where

- ▶ A is a nonempty set.
- ▶ $\mathcal{F}^{\mathcal{A}} = \{f^{\mathcal{A}} \mid f \in \mathcal{F}\}$ is a set of functions on A ; if f has arity m , then $f^{\mathcal{A}} : A^m \rightarrow A$.
- ▶ $\mathcal{R}^{\mathcal{A}} = \{R^{\mathcal{A}} \mid R \in \mathcal{R}\}$ is a set of relations on A ; if R has arity m , then $R^{\mathcal{A}} \subseteq A^m$.
- ▶ $\mathcal{C}^{\mathcal{A}} = \{c^{\mathcal{A}} \in A \mid c \in \mathcal{C}\}$.
- ▶ A is called the **universe** of the structure \mathcal{A} . **Notation:** $A = |\mathcal{A}|$
- ▶ $f^{\mathcal{A}}$ ($R^{\mathcal{A}}$, $c^{\mathcal{A}}$, respectively) is called the **interpretation** of f (R , c , respectively) in \mathcal{A} .

43

$\mathcal{L}_= = (\mathcal{R}, \mathcal{F}, \mathcal{C})$, where

- ▶ $\mathcal{R} = \mathcal{F} = \mathcal{C} = \emptyset$;
- ▶ this language is proper for expressing the properties of equality;
- ▶ $\mathcal{L}_=$ -structures are the nonempty sets.

Examples of formulas:

- equality is symmetric:

$$\forall x\forall y(x = y \rightarrow y = x)$$

- the universe has at least three elements:

$$\exists x\exists y\exists z(\neg(x = y) \wedge \neg(y = z) \wedge \neg(z = x))$$

44

Examples - The language of arithmetics \mathcal{L}_{ar}

$\mathcal{L}_{ar} = (\mathcal{R}, \mathcal{F}, \mathcal{C})$, where

- ▶ $\mathcal{R} = \{<\}; <$ is a binary relation symbol;
- ▶ $\mathcal{F} = \{+, \times, \dot{S}\}$; $+$, \times are binary function symbols and \dot{S} is a unary function symbol;
- ▶ $\mathcal{C} = \{\dot{0}\}$.

We write $\mathcal{L}_{ar} = (<; +, \times, \dot{S}; \dot{0})$ or $\mathcal{L}_{ar} = (<, +, \times, \dot{S}, \dot{0})$.

The natural example of \mathcal{L}_{ar} -structure:

$$\mathcal{N} := (\mathbb{N}, <, +, \cdot, S, 0),$$

where $S : \mathbb{N} \rightarrow \mathbb{N}$, $S(m) = m + 1$ is the successor function. Thus,

$$<^{\mathcal{N}} = <, +^{\mathcal{N}} = +, \times^{\mathcal{N}} = \cdot, \dot{S}^{\mathcal{N}} = S, \dot{0}^{\mathcal{N}} = 0.$$

- Another example of \mathcal{L}_{ar} -structure: $\mathcal{A} = (\{0, 1\}, <, \vee, \wedge, \neg, 1)$.

45

Examples - The language with a binary relation symbol

$\mathcal{L}_R = (\mathcal{R}, \mathcal{F}, \mathcal{C})$, where

- ▶ $\mathcal{R} = \{R\}$; R is a binary relation symbol;
- ▶ $\mathcal{F} = \mathcal{C} = \emptyset$;
- ▶ \mathcal{L} -structures are nonempty sets together with a binary relation.

- ▶ If we are interested in partially ordered sets (A, \leq) , we use the symbol \leq instead of R and we denote the language by \mathcal{L}_{\leq} .
- ▶ If we are interested in strictly ordered sets $(A, <)$, we use the symbol $<$ instead of R and we denote the language by $\mathcal{L}_{<}$.
- ▶ If we are interested in graphs $G = (V, E)$, we use the symbol \dot{E} instead of R and we denote the language by \mathcal{L}_{Graf} .
- ▶ If we are interested in structures (A, \in) , we use the symbol \in instead of R and we denote the language by \mathcal{L}_{\in} .

46

Semantics

Let \mathcal{L} be a first-order language and \mathcal{A} be an \mathcal{L} -structure.

Definition 2.12

An \mathcal{A} -assignment or \mathcal{A} -evaluation is a function $e : V \rightarrow A$.

When the \mathcal{L} -structure \mathcal{A} is clear from the context, we also write simply e is an assignment.

In the following, $e : V \rightarrow A$ is an \mathcal{A} -assignment.

Definition 2.13 (Interpretation of terms)

The **interpretation** $t^{\mathcal{A}}(e) \in A$ of a term t under the \mathcal{A} -assignment e is defined by induction on terms :

- ▶ if $t = x \in V$, then $t^{\mathcal{A}}(e) := e(x)$;
- ▶ if $t = c \in \mathcal{C}$, then $t^{\mathcal{A}}(e) := c^{\mathcal{A}}$;
- ▶ if $t = ft_1 \dots t_m$, then $t^{\mathcal{A}}(e) := f^{\mathcal{A}}(t_1^{\mathcal{A}}(e), \dots, t_m^{\mathcal{A}}(e))$.

47

Semantics

The **interpretation**

$$\varphi^{\mathcal{A}}(e) \in \{0, 1\}$$

of a formula φ under the \mathcal{A} -assignment e is defined by induction on formulas.

$$(s = t)^{\mathcal{A}}(e) = \begin{cases} 1 & \text{if } s^{\mathcal{A}}(e) = t^{\mathcal{A}}(e) \\ 0 & \text{otherwise.} \end{cases}$$

$$(Rt_1 \dots t_m)^{\mathcal{A}}(e) = \begin{cases} 1 & \text{if } R^{\mathcal{A}}(t_1^{\mathcal{A}}(e), \dots, t_m^{\mathcal{A}}(e)) \\ 0 & \text{otherwise.} \end{cases}$$

48

Negation and implication

- ▶ $(\neg\varphi)^{\mathcal{A}}(e) = 1 - \varphi^{\mathcal{A}}(e)$;
- ▶ $(\varphi \rightarrow \psi)^{\mathcal{A}}(e) = \varphi^{\mathcal{A}}(e) \rightarrow \psi^{\mathcal{A}}(e)$, where,

$$\rightarrow: \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\},$$

p	q	$p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Hence,

- ▶ $(\neg\varphi)^{\mathcal{A}}(e) = 1$ iff $\varphi^{\mathcal{A}}(e) = 0$.
- ▶ $(\varphi \rightarrow \psi)^{\mathcal{A}}(e) = 1$ iff $(\varphi^{\mathcal{A}}(e) = 0$ or $\psi^{\mathcal{A}}(e) = 1)$.

Notation

For any variable $x \in V$ and any $a \in A$, we define a new \mathcal{A} -assignment $e_{x \mapsto a} : V \rightarrow A$ by

$$e_{x \mapsto a}(v) = \begin{cases} e(v) & \text{if } v \neq x \\ a & \text{if } v = x. \end{cases}$$

Universal quantifier

$$(\forall x \varphi)^{\mathcal{A}}(e) = \begin{cases} 1 & \text{if } \varphi^{\mathcal{A}}(e_{x \mapsto a}) = 1 \text{ for all } a \in A \\ 0 & \text{otherwise.} \end{cases}$$

Let \mathcal{A} be an \mathcal{L} -structure and $e : V \rightarrow A$ be an \mathcal{A} -assignment.

Definition 2.14

Let φ be a formula. We say that:

- ▶ e **satisfies** φ in \mathcal{A} if $\varphi^{\mathcal{A}}(e) = 1$. **Notation:** $\mathcal{A} \models \varphi[e]$.
- ▶ e **does not satisfy** φ in \mathcal{A} if $\varphi^{\mathcal{A}}(e) = 0$. **Notation:** $\mathcal{A} \not\models \varphi[e]$.

Proposition 2.15

For all formulas φ, ψ and any variable x ,

- (i) $\mathcal{A} \models \neg\varphi[e]$ iff $\mathcal{A} \not\models \varphi[e]$.
- (ii) $\mathcal{A} \models (\varphi \rightarrow \psi)[e]$ iff $(\mathcal{A} \models \varphi[e]$ implies $\mathcal{A} \models \psi[e])$
iff $(\mathcal{A} \not\models \varphi[e]$ or $\mathcal{A} \models \psi[e])$.
- (iii) $\mathcal{A} \models (\forall x \varphi)[e]$ iff for all $a \in A$, $\mathcal{A} \models \varphi[e_{x \mapsto a}]$.

Proposition 2.16

For all formulas φ, ψ and any variable x ,

- (i) $\mathcal{A} \models (\varphi \wedge \psi)[e]$ iff $(\mathcal{A} \models \varphi[e]$ and $\mathcal{A} \models \psi[e])$.
- (ii) $\mathcal{A} \models (\varphi \vee \psi)[e]$ iff $(\mathcal{A} \models \varphi[e]$ or $\mathcal{A} \models \psi[e])$.
- (iii) $\mathcal{A} \models (\varphi \leftrightarrow \psi)[e]$ iff $(\mathcal{A} \models \varphi[e]$ iff $\mathcal{A} \models \psi[e])$.
- (iv) $\mathcal{A} \models (\exists x \varphi)[e]$ iff there exists $a \in A$ s.t. $\mathcal{A} \models \varphi[e_{x \mapsto a}]$.

Let φ be a formula of \mathcal{L} .

Definition 2.17

φ is **satisfiable** if there exists an \mathcal{L} -structure \mathcal{A} and an \mathcal{A} -assignment e such that $\mathcal{A} \models \varphi[e]$.

We also say that (\mathcal{A}, e) is a **model** of φ .

Definition 2.18

φ is **true** in an \mathcal{L} -structure \mathcal{A} if $\mathcal{A} \models \varphi[e]$ for all \mathcal{A} -assignments e .

We also say that \mathcal{A} **satisfies** φ or that \mathcal{A} is a **model** of φ .

Notation: $\mathcal{A} \models \varphi$

Definition 2.19

φ is **universally true** (or **logically valid** or, simply, **valid**) if $\mathcal{A} \models \varphi$ for all \mathcal{L} -structures \mathcal{A} .

Notation: $\models \varphi$

53

Let φ, ψ be formulas of \mathcal{L} .

Definition 2.20

ψ is a **logical consequence** of φ if for all \mathcal{L} -structures \mathcal{A} and all \mathcal{A} -assignments e ,

Notation: $\mathcal{A} \models \varphi[e]$ implies $\mathcal{A} \models \psi[e]$.

Notation: $\varphi \models \psi$

Definition 2.21

φ and ψ are **logically equivalent** or, simply, **equivalent** if for all \mathcal{L} -structures \mathcal{A} and all \mathcal{A} -assignments e ,

$\mathcal{A} \models \varphi[e]$ iff $\mathcal{A} \models \psi[e]$.

Notation: $\varphi \equiv \psi$

Remark

▶ $\varphi \models \psi$ iff $\models \varphi \rightarrow \psi$.

▶ $\varphi \equiv \psi$ iff $(\psi \models \varphi \text{ and } \varphi \models \psi)$ iff $\models \psi \leftrightarrow \varphi$.

54

For all formulas φ, ψ and all variables x, y ,

$$\neg \exists x \varphi \models \forall x \neg \varphi \quad (1)$$

$$\neg \forall x \varphi \models \exists x \neg \varphi \quad (2)$$

$$\forall x (\varphi \wedge \psi) \models \forall x \varphi \wedge \forall x \psi \quad (3)$$

$$\forall x \varphi \vee \forall x \psi \models \forall x (\varphi \vee \psi) \quad (4)$$

$$\exists x (\varphi \wedge \psi) \models \exists x \varphi \wedge \exists x \psi \quad (5)$$

$$\exists x (\varphi \vee \psi) \models \exists x \varphi \vee \exists x \psi \quad (6)$$

$$\forall x (\varphi \rightarrow \psi) \models \forall x \varphi \rightarrow \forall x \psi \quad (7)$$

$$\forall x (\varphi \rightarrow \psi) \models \exists x \varphi \rightarrow \exists x \psi \quad (8)$$

$$\forall x \varphi \models \exists x \varphi \quad (9)$$

55

$$\varphi \models \exists x \varphi \quad (10)$$

$$\forall x \varphi \models \varphi \quad (11)$$

$$\forall x \forall y \varphi \models \forall y \forall x \varphi \quad (12)$$

$$\exists x \exists y \varphi \models \exists y \exists x \varphi \quad (13)$$

$$\exists y \forall x \varphi \models \forall x \exists y \varphi. \quad (14)$$

56

Proposition 2.22

For all terms s, t, u ,

- (i) $\models t = t$;
- (ii) $\models s = t \rightarrow t = s$;
- (iii) $\models s = t \wedge t = u \rightarrow s = u$.

Proposition 2.23

For all $m \geq 1$, $f \in \mathcal{F}_m$, $R \in \mathcal{R}_m$ and all terms $t_i, u_i, i = 1, \dots, m$,

$$\begin{aligned} \models (t_1 = u_1) \wedge \dots \wedge (t_m = u_m) &\rightarrow ft_1 \dots t_m = fu_1 \dots u_m \\ \models (t_1 = u_1) \wedge \dots \wedge (t_m = u_m) &\rightarrow (Rt_1 \dots t_m \leftrightarrow Ru_1 \dots u_m) \end{aligned}$$

57

Proposition 2.24

For any \mathcal{L} -structure \mathcal{A} and any \mathcal{A} -assignments e_1, e_2 ,

- (i) for any term t ,
if $e_1(v) = e_2(v)$ for all variables $v \in \text{Var}(t)$, then
 $t^{\mathcal{A}}(e_1) = t^{\mathcal{A}}(e_2)$.
- (ii) for any formula φ ,
if $e_1(v) = e_2(v)$ for all variables $v \in \text{FV}(\varphi)$, then $\mathcal{A} \models \varphi[e_1]$
iff $\mathcal{A} \models \varphi[e_2]$.

58

Definition 2.25

A formula φ is called a **sentence** if $\text{FV}(\varphi) = \emptyset$, that is φ does not have free variables.

Notation: $\text{Sent}_{\mathcal{L}}$:= the set of sentences of \mathcal{L} .

Proposition 2.26

Let φ be a sentence. For all \mathcal{A} -assignments e_1, e_2 ,

$$\mathcal{A} \models \varphi[e_1] \iff \mathcal{A} \models \varphi[e_2]$$

Definition 2.27

Let φ be a sentence. An \mathcal{L} -structure \mathcal{A} is a **model** of φ if $\mathcal{A} \models \varphi[e]$ for an (any) \mathcal{A} -assignment e . **Notation:** $\mathcal{A} \models \varphi$

59

Let φ be a sentence and Γ be a set of sentences of \mathcal{L} .

Definition 2.28

An \mathcal{L} -structure \mathcal{A} is a **model** of Γ if

$$\mathcal{A} \models \gamma \text{ for all } \gamma \in \Gamma.$$

Notation: $\mathcal{A} \models \Gamma$

Definition 2.29

Γ is **satisfiable** if Γ has a model.

Definition 2.30

φ is a **logical consequence** of Γ if for all \mathcal{L} -structures \mathcal{A} ,

$$\mathcal{A} \models \Gamma \implies \mathcal{A} \models \varphi.$$

Notation: $\Gamma \models \varphi$

60

Examples

For all $n \geq 2$, we denote by $\exists^{\geq n}$ the following sentence:

$$\exists x_1 \dots \exists x_n (\neg(x_1 = x_2) \wedge \neg(x_1 = x_3) \wedge \dots \wedge \neg(x_{n-1} = x_n)),$$

written in a more compact way as follows:

$$\exists^{\geq n} = \exists x_1 \dots \exists x_n \left(\bigwedge_{1 \leq i < j \leq n} \neg(x_i = x_j) \right).$$

Proposition 2.31

For any \mathcal{L} -structure \mathcal{A} and any $n \geq 2$,

$$\mathcal{A} \models \exists^{\geq n} \iff \mathcal{A} \text{ has at least } n \text{ elements.}$$

Proof: Easy exercise.

61

Examples

Notations

- ▶ For uniformity, let $\exists^{\geq 1} := \exists x(x = x)$.
- ▶ Denote $\exists^{\leq n} := \neg \exists^{\geq n+1}$ and $\exists^{=n} := \exists^{\leq n} \wedge \exists^{\geq n}$

Proposition 2.32

For any \mathcal{L} -structure \mathcal{A} and any $n \geq 1$,

$$\begin{aligned} \mathcal{A} \models \exists^{\leq n} &\iff \mathcal{A} \text{ has at most } n \text{ elements} \\ \mathcal{A} \models \exists^{=n} &\iff \mathcal{A} \text{ has exactly } n \text{ elements.} \end{aligned}$$

Proof: Easy exercise.

Proposition 2.33

Let $\Gamma := \{\exists^{\geq n} \mid n \geq 1\}$. Then for any \mathcal{L} -structure \mathcal{A} ,

$$\mathcal{A} \models \Gamma \iff \mathcal{A} \text{ is an infinite set.}$$

Proof: Easy exercise.

62

Substitution

Let x be a variable of \mathcal{L} and u be a term of \mathcal{L} .

Definition 2.34

For any term t of \mathcal{L} , we define

$$t_x(u) := \text{the expression obtained from } t \text{ by replacing all occurrences of } x \text{ with } u.$$

Proposition 2.35

For any term t of \mathcal{L} , $t_x(u)$ is a term of \mathcal{L} .

63

Substitution

- ▶ We would like to define, similarly, $\varphi_x(u)$ as the expression obtained from φ by replacing all free occurrences of x in φ with u .
- ▶ We expect that the following natural properties of substitution are true:

$$\models \forall x \varphi \rightarrow \varphi_x(u) \quad \text{and} \quad \models \varphi_x(u) \rightarrow \exists x \varphi.$$

As the following example shows, there are problems with this definition.

Let $\varphi := \exists y \neg(x = y)$ and $u := y$. Then $\varphi_x(u) = \exists y \neg(y = y)$.

Avem

- ▶ For any \mathcal{L} -structure \mathcal{A} with $|A| \geq 2$, $\mathcal{A} \models \forall x \varphi$.
- ▶ $\varphi_x(u)$ is not satisfiable.

64

Substitution

Let x be a variable, u a term and φ a formula.

Definition 2.36

We say that x is **free for u** in φ or that u is **substitutable for x** in φ if for any variable y that occurs in u , no subformula of φ of the form $\forall y\psi$ contains free occurrences of x .

Remark

x is free for u in φ in any of the following cases:

- ▶ u does not contain variables;
- ▶ φ does not contain variables that occur in u ;
- ▶ no variable from u occurs bound in φ ;
- ▶ x does not occur in φ ;
- ▶ φ does not contain free occurrences of x .

65

Substitution

Let x be a variable, u a term and φ be a formula such that x is free for u in φ .

Definition 2.37

$\varphi_x(u) :=$ the expression obtained from φ by replacing all free occurrences of x in φ with u .

We say that $\varphi_x(u)$ is a **free substitution**.

Proposition 2.38

$\varphi_x(u)$ is a formula of \mathcal{L} .

66

Substitution

Free substitution rules out the problems mentioned above, it behaves as expected.

Proposition 2.39

Let φ be a formula and x be a variable. For any term u substitutable for x in φ ,

$$\models \forall x\varphi \rightarrow \varphi_x(u) \quad \text{and} \quad \models \varphi_x(u) \rightarrow \exists x\varphi.$$

67

Prenex normal form

Definition 2.40

A formula that does not contain quantifiers is called **quantifier-free**.

Definition 2.41

A formula φ is in **prenex normal form** if

$$\varphi = Q_1x_1Q_2x_2 \dots Q_nx_n\psi,$$

where $n \in \mathbb{N}$, $Q_1, \dots, Q_n \in \{\forall, \exists\}$, x_1, \dots, x_n are variables and ψ is a quantifier-free formula. $Q_1x_1Q_2x_2 \dots Q_nx_n$ is the **prefix** of φ and ψ is called the **matrix** of φ .

Any quantifier-free formula is in prenex normal form, as one can take $n = 0$ in the above definition.

68

Prenex normal form

Examples of formulas in prenex normal form:

- ▶ **universal** formulas: $\varphi = \forall x_1 \forall x_2 \dots \forall x_n \psi$, where ψ is quantifier-free
- ▶ **existential** formulas: $\varphi = \exists x_1 \exists x_2 \dots \exists x_n \psi$, where ψ is quantifier-free
- ▶ **$\forall\exists$** -formulas: $\varphi = \forall x_1 \forall x_2 \dots \forall x_n \exists y_1 \exists y_2 \dots \exists y_k \psi$, where ψ is quantifier-free
- ▶ **$\forall\exists\forall$** -formulas: $\varphi = \forall x_1 \dots \forall x_n \exists y_1 \dots \exists y_k \forall z_1 \dots \forall z_p \psi$, where ψ is quantifier-free

Theorem 2.42 (Prenex normal form theorem)

For any formula φ there exists a formula φ^* in prenex normal form such that $\varphi \vDash \varphi^*$ and $FV(\varphi) = FV(\varphi^*)$.
 φ^* is called a **prenex normal form** of φ .

69

Prenex normal form

Proposition 2.43

For all formulas φ, ψ and any variable $x \notin FV(\varphi)$,

$$\forall x(\varphi \wedge \psi) \vDash \varphi \wedge \forall x\psi \quad (15)$$

$$\forall x(\varphi \vee \psi) \vDash \varphi \vee \forall x\psi \quad (16)$$

$$\exists x(\varphi \wedge \psi) \vDash \varphi \wedge \exists x\psi \quad (17)$$

$$\exists x(\varphi \vee \psi) \vDash \varphi \vee \exists x\psi \quad (18)$$

$$\forall x(\varphi \rightarrow \psi) \vDash \varphi \rightarrow \forall x\psi \quad (19)$$

$$\exists x(\varphi \rightarrow \psi) \vDash \varphi \rightarrow \exists x\psi \quad (20)$$

$$\forall x(\psi \rightarrow \varphi) \vDash \exists x\psi \rightarrow \varphi \quad (21)$$

$$\exists x(\psi \rightarrow \varphi) \vDash \forall x\psi \rightarrow \varphi \quad (22)$$

Proposition 2.44

For any formula φ , distinct variables x and y such that y does not occur in φ ,

$$\exists x\varphi \vDash \exists y\varphi_x(y) \quad \text{and} \quad \forall x\varphi \vDash \forall y\varphi_x(y).$$

70

Prenex normal form

Let \mathcal{L} be a first-order language containing

- ▶ two unary relation symbols R, S and two binary relation symbols P, Q ;
- ▶ a unary function symbol f and a binary function symbol g ;
- ▶ two constant symbols c, d .

Example

Find a prenex normal form of the formula

$$\varphi := \exists y(g(y, z) = c) \wedge \neg \exists x(f(x) = d)$$

We have that

$$\varphi \vDash \exists y(g(y, z) = c \wedge \neg \exists x(f(x) = d))$$

$$\vDash \exists y(g(y, z) = c \wedge \forall x \neg(f(x) = d))$$

$$\vDash \exists y \forall x (g(y, z) = c \wedge \neg(f(x) = d))$$

Thus, $\varphi^* = \exists y \forall x (g(y, z) = c \wedge \neg(f(x) = d))$ is a prenex normal form of φ .

71

Prenex normal form

Example

Find a prenex normal form of the formula

$$\varphi := \neg \forall y(S(y) \rightarrow \exists z R(z)) \wedge \forall x(\forall y P(x, y) \rightarrow f(x) = d).$$

$$\varphi \vDash \exists y \neg(S(y) \rightarrow \exists z R(z)) \wedge \forall x(\forall y P(x, y) \rightarrow f(x) = d)$$

$$\vDash \exists y \neg \exists z(S(y) \rightarrow R(z)) \wedge \forall x(\forall y P(x, y) \rightarrow f(x) = d)$$

$$\vDash \exists y \neg \exists z(S(y) \rightarrow R(z)) \wedge \forall x \exists y(P(x, y) \rightarrow f(x) = d)$$

$$\vDash \exists y \forall z \neg(S(y) \rightarrow R(z)) \wedge \forall x \exists y(P(x, y) \rightarrow f(x) = d)$$

$$\vDash \exists y \forall z \forall x(\neg(S(y) \rightarrow R(z)) \wedge \exists y(P(x, y) \rightarrow f(x) = d))$$

$$\vDash \exists y \forall z \forall x(\neg(S(y) \rightarrow R(z)) \wedge \exists v(P(x, v) \rightarrow f(x) = d))$$

$$\vDash \exists y \forall z \forall x(\neg(S(y) \rightarrow R(z)) \wedge \exists v(P(x, v) \rightarrow f(x) = d))$$

$$\vDash \exists y \forall z \forall x \exists v(\neg(S(y) \rightarrow R(z)) \wedge (P(x, v) \rightarrow f(x) = d))$$

$\varphi^* = \exists y \forall z \forall x \exists v(\neg(S(y) \rightarrow R(z)) \wedge (P(x, v) \rightarrow f(x) = d))$ is a prenex normal form of φ .

72

Definition 2.45

The set $\text{LogAx}_{\mathcal{L}} \subseteq \text{Form}_{\mathcal{L}}$ of **logical axioms** of \mathcal{L} consists of:

- (i) the axioms of the propositional logic LP.
- (ii) formulas of the form

$$t = t, \quad s = t \rightarrow t = s, \quad s = t \wedge t = u \rightarrow s = u,$$
 for any terms s, t, u .
- (iii) formulas of the form

$$t_1 = u_1 \wedge \dots \wedge t_m = u_m \rightarrow ft_1 \dots t_m = fu_1 \dots u_m,$$

$$t_1 = u_1 \wedge \dots \wedge t_m = u_m \rightarrow (Rt_1 \dots t_m \leftrightarrow Ru_1 \dots u_m),$$
 for any $m \geq 1$, $f \in \mathcal{F}_m$, $R \in \mathcal{R}_m$ and any terms s_i, t_i ($i = 1, \dots, m$).
- (iv) formulas of the form

$$\varphi_x(t) \rightarrow \exists x\varphi,$$
 where $\varphi_x(t)$ is a free substitution (**\exists -axioms**).

73

Definition 2.46

The **deduction rules** (or **inference rules**) are the following: for any formulas φ, ψ ,

- (i) from φ and $\varphi \rightarrow \psi$ infer ψ (**modus ponens** or (**MP**)):

$$\frac{\varphi, \varphi \rightarrow \psi}{\psi}$$

- (ii) if $x \notin \text{FV}(\psi)$, then from $\varphi \rightarrow \psi$ infer $\exists x\varphi \rightarrow \psi$ (**\exists -introduction**):

$$\frac{\varphi \rightarrow \psi}{\exists x\varphi \rightarrow \psi} \quad \text{if } x \notin \text{FV}(\psi).$$

74

Let Γ be a set of formulas of \mathcal{L} .

Definition 2.47

The **Γ -theorems** of \mathcal{L} are the formulas defined as follows:

- ($\Gamma 0$) Every logical axiom is a Γ -theorem.
- ($\Gamma 1$) Every formula of Γ is a Γ -theorem.
- ($\Gamma 2$) If φ and $\varphi \rightarrow \psi$ are Γ -theorems, then ψ is a Γ -theorem.
- ($\Gamma 3$) If $\varphi \rightarrow \psi$ is a Γ -theorem and $x \notin \text{FV}(\psi)$, then $\exists x\varphi \rightarrow \psi$ is a Γ -theorem.
- ($\Gamma 4$) Only the formulas obtained by applying rules ($\Gamma 0$), ($\Gamma 1$), ($\Gamma 2$) and ($\Gamma 3$) are Γ -theorems.

If φ is a Γ -theorem, then we also say that φ is **deduced from the hypotheses** Γ .

75

Notations

- $\Gamma \vdash_{\mathcal{L}} \varphi$:= φ is a Γ -theorem
- $\vdash_{\mathcal{L}} \varphi$:= $\emptyset \vdash_{\mathcal{L}} \varphi$

Definition 2.48

A formula φ is called a **(logical) theorem** of \mathcal{L} if $\vdash_{\mathcal{L}} \varphi$.

Convention

When \mathcal{L} is clear from the context, we write $\Gamma \vdash \varphi$, $\vdash \varphi$, etc..

76

Definition 2.49

A Γ -proof (or *proof from the hypotheses Γ*) of \mathcal{L} is a sequence of formulas $\theta_1, \dots, \theta_n$ such that for all $i \in \{1, \dots, n\}$, one of the following holds:

- (i) θ_i is an axiom;
- (ii) $\theta_i \in \Gamma$;
- (iii) there exist $k, j < i$ such that $\theta_k = \theta_j \rightarrow \theta_i$;
- (iv) there exists $j < i$ such that

$$\theta_j = \varphi \rightarrow \psi \text{ and } \theta_i = \exists x\varphi \rightarrow \psi,$$
 where φ, ψ are formulas and $x \notin FV(\psi)$.

A \emptyset -proof is called simply a *proof*.

Definition 2.50

Let φ be a formula. A Γ -proof of φ or a *proof of φ from the hypotheses Γ* is a Γ -proof $\theta_1, \dots, \theta_n$ such that $\theta_n = \varphi$.

Proposition 2.51

Let Γ be a set of formulas. For any formula φ ,

$\Gamma \vdash \varphi$ iff there exists a Γ -proof of φ .

Proposition 2.52

For any formula φ and variable x ,

$$\Gamma \vdash \varphi \iff \Gamma \vdash \forall x\varphi.$$

Definition 2.53

Let φ be a formula with $FV(\varphi) = \{x_1, \dots, x_n\}$. The *universal closure* of φ is the sentence

$$\overline{\forall\varphi} := \forall x_1 \dots \forall x_n \varphi.$$

Notation 2.54

$$\overline{\forall\Gamma} := \{\overline{\forall\psi} \mid \psi \in \Gamma\}.$$

Proposition 2.55

For any formula φ ,

$$\Gamma \vdash \varphi \iff \Gamma \vdash \overline{\forall\varphi} \iff \overline{\forall\Gamma} \vdash \varphi \iff \overline{\forall\Gamma} \vdash \overline{\forall\varphi}.$$

Theorem 2.56 (Completeness Theorem)

For any set of sentences Γ and any sentence φ ,

$$\Gamma \vdash \varphi \iff \Gamma \models \varphi.$$

- ▶ The Completeness Theorem was proved by Gödel in 1929 in his PhD thesis.
- ▶ Henkin gave in 1949 a simplified proof.

Intelligent Agents

Textbook:

Michael Wooldridge, *An Introduction to MultiAgent Systems*,
Second Edition, John Wiley & Sons, 2009

We also use

Lecture slides/handouts, made available by Michael Wooldridge
[here](#)

81

Agents

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

Definition

An **agent** is a (computer) system that is **situated** in some **environment**, and that is capable of **autonomous action** in the environment in order to meet its delegated objectives.

The agent figures out what needs to be done to satisfy design objectives, rather than constantly being told by its user or owner.

82

Agents

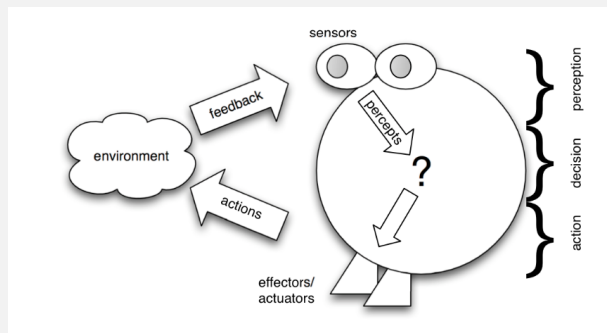


Figure 1: An agent in its environment

- ▶ Figure 1 gives 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**.

83

- ▶ Usually, an agent will **not** have **complete control** over its environment.
- ▶ It will have at best **partial control**, in that it can **influence** it.
- ▶ From the point of view of the agent, this means that the **same action** performed **twice** in apparently identical circumstances might appear to have entirely **different effects**, and in particular, it may **fail** to have the desired effect.
- ▶ Thus agents in all but the most trivial of environments must be prepared for the possibility of **failure**.

84

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.

85

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.

86

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 $\alpha_0, \alpha_1, \dots$

87

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.

88

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

Definition 3.1

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:

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

or

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

89

Notation 3.2

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

90

Definition 3.3

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 :

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

91

Definition 3.4

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 τ 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.

92

We introduce a model of the agents that inhabit systems.

Definition 3.5

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 3.6

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

Definition 3.7

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 $k \geq 0$:
 - $x_{2k+1} = Ag(x_0, \dots, x_{2k})$ 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 $Ag(r) = null$;
 - ▶ if $x_n \in Ac$, then $\tau(r) = \emptyset$.

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

Notation 3.8

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

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 $j = 1, \dots, u - 1$,

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

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

- ▶ $\tau(r) = \emptyset$.

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 3.9

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.

97

- ▶ 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 3.10

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

98

A **thermostat** is a very simple example of a **purely reactive agent**.

- ▶ 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.

99

- ▶ Let us denote
 - $e_1 :=$ temperature too low, $e_2 :=$ temperature OK,
 - $\alpha_1 :=$ heating on, $\alpha_2 :=$ 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:

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

100

Intelligent agents

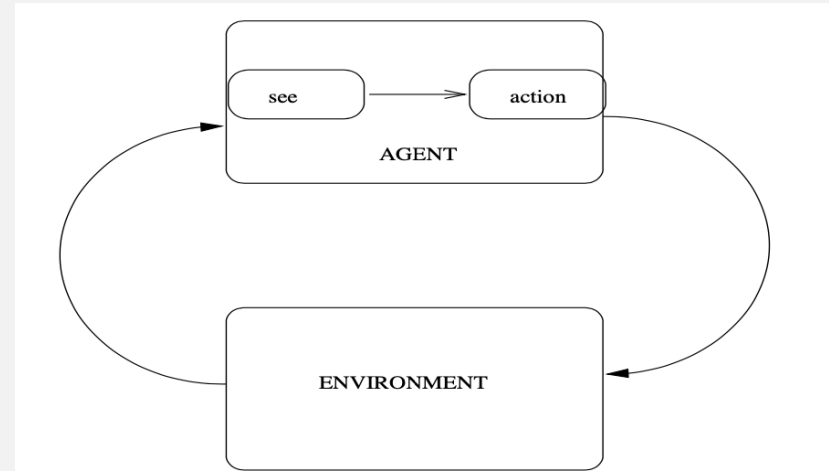
This view of agents is too abstract. It does not help us to **construct** them, since it gives us no clues about how to design the decision function **action**.

- ▶ 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.

101

Perception

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



102

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 3.11

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 :

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

103

Perception

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 3.12

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 3.13

\equiv is an equivalence relation on E .

104

Perception

- ▶ \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.

105

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.

106

Perception - an example

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

$p_1 :=$ temperature OK, $p_2 :=$ temperature too low.

The perception function see is defined as follows:

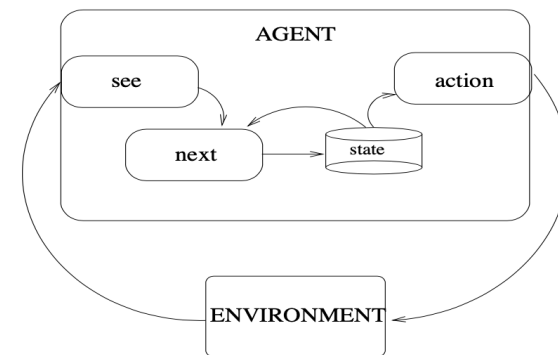
$$see : E \rightarrow Per, \quad see(e) = \begin{cases} p_1 & \text{if } e = e_1 \text{ or } e = e_2, \\ p_2 & \text{if } e = e_3 \text{ or } e = e_4. \end{cases}$$

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

107

Agents with state

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



These agents have some **internal data structure**, which is typically used to record information about the environment state and history.

108

Agents with state

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

Definition 3.14

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 3.15

The function *next* is defined as follows:

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

109

Agents with state

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 $\text{see}(e)$.
- ▶ The internal state of the agent is then updated to $i_1 := \text{next}(i_0, \text{see}(e))$.
- ▶ The action selected by the agent is $\alpha := \text{action}(i_1)$.
- ▶ The agent performs action α .
- ▶ The agent enters another cycle: perceives the world via *see*, updates its state via *next*, and chooses an action to perform via *action*.

110

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 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: we have to think about exactly how the task will be carried out ourselves; if unforeseen circumstances arise, the agent executing the task will be unable to respond accordingly.

111

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.

112

Definition 3.16

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.

113

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

Definition 3.17

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.

114

Tileworld was proposed as an experimental environment for evaluating agent architectures in
Martha E. Pollock, Marc Ringuette, Introducing the Tileworld: Experimentally Evaluating Agent Architectures, AAAI-90 Proceedings, 1990

- ▶ 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**.

115

- ▶ 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.

116

Definition 3.18

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.

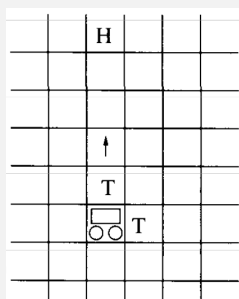


Figure 2: Tileworld example

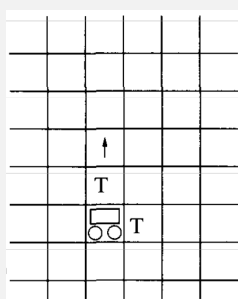


Figure 3: Tileworld example

Example 3.19

Suppose an agent is pushing a tile to a hole (Figure 2), when this hole disappears (Figure 3). The agent should recognize this change in the environment, and modify its behaviour appropriately.

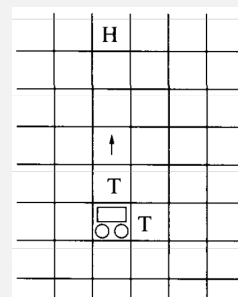


Figure 4: Tileworld example

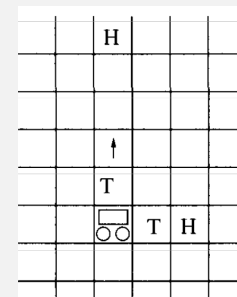


Figure 5: Tileworld example

Example 3.20

Suppose an agent is pushing a tile to a hole (Figure 4), when a hole appears to the right of the agent (Figure 5). 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.

Expected utility

Let us denote $P(r | Ag, Env)$ the probability that run r occurs when agent Ag is placed in environment Env .

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

Definition 3.22

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

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

121

Expected utility - an example

Consider the environment (E, e_0, τ) 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:

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

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

122

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 :

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

► for the agent Ag_2 :

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

123

Expected utility - an example

Assume the utility function u is defined as follows:

$$u(r_1) = 8, \quad u(r_2) = 11, \\ u(r_3) = 70, \quad u(r_4) = 9, \quad u(r_5) = 10.$$

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

$$EU(Ag_1, Env) = u(r_1)P(r_1 | Ag_1, Env) + u(r_2)P(r_2 | Ag_1, Env) \\ = 8 \times 0.4 + 11 \times 0.6 = 9.6$$

$$EU(Ag_2, Env) = u(r_3)P(r_3 | Ag_2, Env) + u(r_4)P(r_4 | Ag_2, Env) \\ + u(r_5)P(r_5 | Ag_2, Env) \\ = 70 \times 0.1 + 9 \times 0.2 + 10 \times 0.7 = 15.7$$

124

Optimal agents

Notation 3.23

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

Definition 3.24

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

125

Bounded optimal agents

Suppose m is a particular computer.

Notation 3.25

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 3.26

A **bounded optimal agent** in an environment Env , with respect to m , is an agent $Ag_{bopt} \in AG_m$ 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.

126

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 3.27

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

127

Task environment

Definition 3.28

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

Notation 3.29

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 Ψ).

128

Task environment

Notation 3.30

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

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

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

The pessimistic definition:

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

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

129

Task environment

The optimistic definition:

We say that an agent Ag **succeeds** in task environment (Env, Ψ) 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, Ψ) is defined as the probability $P(\Psi|Ag, Env)$ that the predicate task specification Ψ is satisfied by the agent in the environment Env .

Remark 3.31

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

130

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** φ .
- ▶ Maintenance tasks are those of the form **maintain state of affairs** φ .

131

Achievement tasks

Definition 3.32

The task environment (Env, Ψ) 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, Ψ) is an **achievement task environment**.

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

Notation 3.33

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).

132

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.

133

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 3.34

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

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

We also say that (Env, Ψ) is a **maintenance task environment**.

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

Notation 3.35

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

134

Maintenance tasks

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 .

135

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 .

136

- ▶ 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.

137

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 3.36

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.

138

Definition 3.37

An agent synthesis algorithm syn is

- ▶ **sound** if for any task environment $(Env, \Psi) \in TE$,
 $\text{syn}((Env, \Psi)) = Ag$ implies $\mathcal{R}(Ag, Env) = \mathcal{R}_\Psi(Ag, Env)$.
- ▶ **complete** if for any $(Env, \Psi) \in TE$,
(there exists an agent Ag s.t. $\mathcal{R}(Ag, Env) = \mathcal{R}_\Psi(Ag, Env)$)
implies $\text{syn}((Env, \Psi)) \neq \perp$.
- ▶ 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.

139

- ▶ 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.

140

Agent architectures

Textbook:

Michael Wooldridge, [An Introduction to MultiAgent Systems](#),
Second Edition, John Wiley & Sons, 2009

141

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.

142

Agent architectures

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.

143

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.

144

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

- ▶ An **internal state** of such an agent is a **database** of formulas of **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.

145

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_{α} for every action $\alpha \in Ac$. For simplicity, we write α instead of c_{α} .

- ▶ 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, $I = \mathcal{D}$.

146

- ▶ We fix a set $\Sigma \subseteq Form_{\mathcal{L}}$ 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 α 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 6.

147

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 α
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 α
11. end-if
12. end-for
13. return $null$
14. end function $action$

Figure 6: Agent selection as theorem proving.

148

Agents as theorem provers

- ▶ In lines 3-7, the agent takes each of its possible actions α in turn, and attempts to prove the formula $Do(\alpha)$ from its database DB (passed as a parameter to the function) using its set Σ of deduction formulas. If the agent succeeds in proving $Do(\alpha)$, then α 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.

149

Agents as theorem provers

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.

150

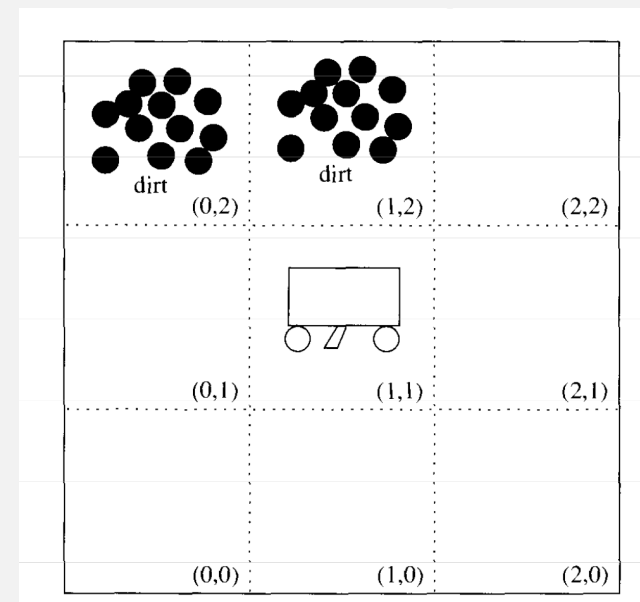
Agents as theorem provers - example

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 × 3 grid**, and the agent always **starts in grid square (0, 0) facing north**.

151

Agents as theorem provers - example



152

- ▶ 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.

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 $i, j \in \{0, 1, 2\}$.

- ▶ 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.

$$old(DB) = DB \cap \Delta,$$

where

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

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

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

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

The *next* function is defined as follows:

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

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).

157

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 $(0, 0)$.

158

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.

159

Modal Logics

Textbook:

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

160

Definition 4.1

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 ;
- ▶ parentheses: $(,)$;
- ▶ the modal operator \Box (**box**).

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

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

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

Definition 4.2

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

- (F0) Every atomic proposition is a formula.
- (F2) If φ is a formula, then $(\neg\varphi)$ is a formula.
- (F3) If φ and ψ are formulas, then $(\varphi \rightarrow \psi)$ is a formula.
- (F4) If φ 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 $Form(ML_0)$.

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

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

Derived connectives

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

$$\begin{aligned} \varphi \vee \psi &:= ((\neg\varphi) \rightarrow \psi) & \varphi \wedge \psi &:= \neg(\varphi \rightarrow (\neg\psi)) \\ \varphi \leftrightarrow \psi &:= ((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)) \\ \top &:= p \rightarrow p, \text{ where } p \in PROP, & \perp &:= \neg\top \end{aligned}$$

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 φ .

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,

- ▶ $\Box\varphi$ is read as **is necessarily φ** .
- ▶ $\Diamond\varphi$ means **it is not necessary that not φ** , that is **it is possible the case that φ** .

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.

165

Definition 4.3

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 4.4

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

166

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

Definition 4.5

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 .

167

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:

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

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 $Rwu_1, Ru_1u_2, \dots, Ru_{n-1}v$.

168

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 4.6

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.

Interpretation using agents

Rwv holds iff the agent considers the world v possible according to the informations available in the world w . We think of R as a **possibility** relation, as R defines worlds that are considered possible by the agent.

Definition 4.7

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.

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 \mathcal{F} 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

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

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 4.8

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

formula φ 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 $p \in PROP$

$\mathcal{M}, w \Vdash \neg\varphi$ 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 φ at w , we write $\mathcal{M}, w \not\Vdash \varphi$ and we say that φ is false in \mathcal{M} at state w .

It follows from Definition 4.8 that for every state $w \in W$,

► $\mathcal{M}, w \Vdash \neg\varphi$ iff $\mathcal{M}, w \not\Vdash \varphi$.

Notation

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

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

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

Proposition 4.9

For every formulas φ, ψ ,

$\mathcal{M}, w \Vdash \varphi \vee \psi$ 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 4.10

For every formula φ ,

$\mathcal{M}, w \Vdash \Diamond\varphi$ 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 4.11

For every $n \geq 1$ and every formula φ , 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 $v \in W$ s.t. $R^n wv$ and $\mathcal{M}, v \Vdash \varphi$

$\mathcal{M}, w \Vdash \Box^n\varphi$ iff for every $v \in W$, $R^n wv$ implies $\mathcal{M}, v \Vdash \varphi$.

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

Definition 4.12

- ▶ A formula φ 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 φ is **satisfiable** in \mathcal{M} if there exists a state $w \in W$ such that $\mathcal{M}, w \Vdash \varphi$.

Definition 4.13

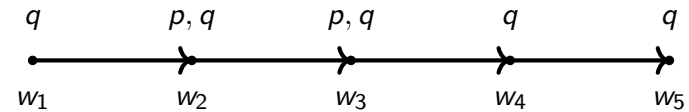
Let Σ be a set of formulas.

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

A model $\mathcal{M} = (W, R, V)$ is represented as a labeled directed graph:

- ▶ the nodes of the graph are the states of the model;
- ▶ the label of each node $w \in W$ describes which atomic propositions are true at state w ;
- ▶ there exists an edge from node w to node v iff Rwv holds.

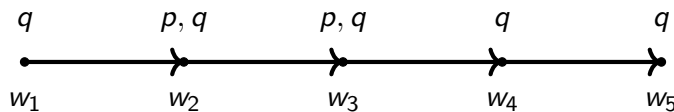
Example



We know that $PROP = \{p, q, r\}$. Then $\mathcal{M} = (W, R, V)$, where $W = \{w_1, w_2, w_3, w_4, w_5\}$; $Rw_i w_j$ iff $j = i + 1$; $V(p) = \{w_2, w_3\}$, $V(q) = \{w_1, w_2, w_3, w_4, w_5\}$ and $V(r) = \emptyset$.

Example

Let $\mathcal{M} = (W, R, V)$ be the model represented by:

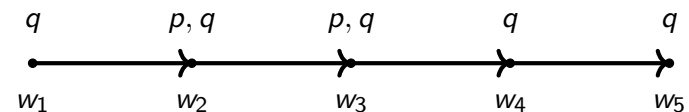


- (i) $\mathcal{M}, w_1 \Vdash \Diamond \Box p$.
- (ii) $\mathcal{M}, w_1 \not\Vdash \Diamond \Box p \rightarrow p$.
- (iii) $\mathcal{M}, w_2 \Vdash \Diamond(p \wedge \neg r)$.
- (iv) $\mathcal{M}, w_1 \Vdash q \wedge \Diamond(q \wedge \Diamond(q \wedge \Diamond(q \wedge \Diamond q)))$.
- (v) $\mathcal{M}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

Proof: (i) $\mathcal{M}, w_1 \Vdash \Diamond \Box p$ iff there exists $v \in W$ such that $Rw_1 v$ and $\mathcal{M}, v \Vdash \Box p$. Take $v := w_2$. As $Rw_1 w_2$, it remains to prove that $\mathcal{M}, w_2 \Vdash \Box p$. We have that $\mathcal{M}, w_2 \Vdash \Box p \iff$ for all $u \in W$, $Rw_2 u$ implies $\mathcal{M}, u \Vdash p \iff \mathcal{M}, w_3 \Vdash p$ (as w_3 is the unique $u \in W$ s.t. $Rw_2 u$) $\iff w_3 \in V(p)$, which is true.

Example

Let $\mathcal{M} = (W, R, V)$ be the model represented by:



Proof: (ii) We have that $\mathcal{M}, w_1 \not\Vdash \Diamond \Box p \rightarrow p \iff \mathcal{M}, w_1 \Vdash \Diamond \Box p$ and $\mathcal{M}, w_1 \not\Vdash p$. Apply (i) and the fact that $w_1 \notin V(p)$.
(iii), (iv) Exercise.
(v) Let $w \in W$ be arbitrary. Then $\mathcal{M}, w \Vdash \Box q \iff$ for all $v \in W$, Rwv implies $\mathcal{M}, v \Vdash q \iff$ for all $v \in W$, Rwv implies $v \in V(q)$, which is true, as $V(q) = W$.

The notion of satisfaction is **internal** and **local**. We evaluate formulas **inside** models, at some particular state w (the **current state**). Modal operators \diamond, \square work locally: we verify the truth of φ **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 4.14

Let \mathcal{F} be a frame and φ be a formula.

- ▶ φ is **valid at a state** w in \mathcal{F} if φ is true at w in every model $\mathcal{M} = (\mathcal{F}, V)$ based on \mathcal{F} .
- ▶ φ 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 4.15

If $\varphi \vee \psi$ is true in a model \mathcal{M} at w , then φ is true in \mathcal{M} at w or ψ is true in \mathcal{M} at w (by Proposition 4.9).

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

Definition 4.16

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

- ▶ φ is **true in \mathbf{M}** if it is true in every model in \mathbf{M} .
Notation: $\mathbf{M} \Vdash \varphi$
- ▶ φ is **valid in \mathbf{F}** if it is valid in every frame in \mathbf{F} .
Notation: $\mathbf{F} \Vdash \varphi$

Definition 4.17

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 4.18

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

Proof: Let $\mathcal{F} = (W, R)$ be an arbitrary frame, w a state in \mathcal{F} and $\mathcal{M} = (\mathcal{F}, V)$ be a model based on \mathcal{F} . We have to show that

$$\mathcal{M}, w \Vdash \Diamond(p \vee q) \rightarrow (\Diamond p \vee \Diamond q).$$

Suppose that $\mathcal{M}, w \Vdash \Diamond(p \vee q)$. Then there exists $v \in W$ such that Rwv and $\mathcal{M}, v \Vdash p \vee q$. We have two cases:

- ▶ $\mathcal{M}, v \Vdash p$. Then $\mathcal{M}, w \Vdash \Diamond p$, so $\mathcal{M}, w \Vdash \Diamond p \vee \Diamond q$.
- ▶ $\mathcal{M}, v \Vdash q$. Then $\mathcal{M}, w \Vdash \Diamond q$, so $\mathcal{M}, w \Vdash \Diamond p \vee \Diamond q$.

We let as an exercise to prove that $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$ is valid in the class of all frames. □

Example 4.19

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

Proof: We have to find a frame $\mathcal{F} = (W, R)$, a state w in \mathcal{F} and a model $\mathcal{M} = (\mathcal{F}, V)$ such that

$$\mathcal{M}, w \not\Vdash \Box p \rightarrow \Box \Box p.$$

Consider the following frame: $\mathcal{F} = (W, R)$, where

$$W = \{0, 1, 2\}, \quad R = \{(0, 1), (1, 2)\}$$

and take a valuation V such that $V(p) = \{1\}$. Then $\mathcal{M}, 0 \Vdash \Box p$, since 1 is the only state R -accessible from 0 and $\mathcal{M}, 1 \Vdash p$, as $1 \in V(p)$.

On the other hand, $\mathcal{M}, 0 \not\Vdash \Box \Box p$, since $R^2 0 2$ and $\mathcal{M}, 2 \not\Vdash p$, as $2 \notin V(p)$. □

Definition 4.20

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

Example 4.21

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

Proof: Let $\mathcal{F} = (W, R)$ be a transitive frame, w a state in \mathcal{F} and $\mathcal{M} = (\mathcal{F}, V)$ be a model based on \mathcal{F} . Assume that $\mathcal{M}, w \Vdash \Box p$. Then for all $v \in W$,

$$(*) \quad R w v \text{ implies } \mathcal{M}, v \Vdash p.$$

Let us prove that $\mathcal{M}, w \Vdash \Box \Box p$. Let $u, u' \in W$ be such that $R w u'$ and $R u' u$. We have to prove that $\mathcal{M}, u \Vdash p$. Since R is transitive, it follows that $R w u$. Applying (*) with $v := u$ we get that $\mathcal{M}, u \Vdash p$. □

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 **S**.

Let **S** be a class of **structures** (frames or models) for ML_0 .
 If **S** is a class of models, then a model **from S** is simply an element \mathcal{M} of **S**. If **S** is a class of frames, then a model **from S** is a model based on a frame in **S**.

Definition 4.22

Let Σ be a set of formulas and φ be a formula. We say that φ is a **semantic consequence of Σ over S** if for all models \mathcal{M} from **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 Σ is true at a state of the model, then φ must be true **at the same state**.

Remark 4.23

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

Example 4.24

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

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

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

Definition 4.25

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

- ▶ Λ contains the following **axioms**:

$$\begin{aligned} (\text{Taut}) \quad & \text{all propositional tautologies,} \\ (\text{K}) \quad & \Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi), \end{aligned}$$

where φ, ψ are formulas of ML_0 .

- ▶ Λ is closed under the following deduction rules:

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

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

$$\text{generalization or necessitation (GEN): } \frac{\varphi}{\Box\varphi}.$$

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

Normal modal logics - tautologies

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) $\varphi \rightarrow (\psi \rightarrow \varphi)$
- (A2) $(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$
- (A3) $(\neg\psi \rightarrow \neg\varphi) \rightarrow (\varphi \rightarrow \psi)$.

Proposition 4.26

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

Remark 4.27

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

193

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 4.18,

Proposition 4.28

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

194

Normal modal logics

Theorem 4.29

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

Lemma 4.30

- ▶ 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 4.31

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

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

195

K

Definition 4.32

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:

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

Definition 4.33

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

If φ has a K-proof, we say that φ is **K-provable**.

Notation: $\vdash_{\mathbf{K}} \varphi$.

Theorem 4.34

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

196

**Definition 4.35**

Let $\varphi, \psi_1, \dots, \psi_n$ ($n \geq 1$) be formulas. We say that φ is **deducible in propositional logic** from ψ_1, \dots, ψ_n if

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

Lemma 4.36

Let $\varphi, \psi_1, \dots, \psi_n$ ($n \geq 1$) be formulas. The following are equivalent:

- ▶ φ is deducible in propositional logic from ψ_1, \dots, ψ_n .
- ▶ $\psi_1 \rightarrow (\psi_2 \rightarrow \dots \rightarrow (\psi_n \rightarrow \varphi))$ is a tautology.

Proof: Use the fact that

$$(\psi_1 \wedge \dots \wedge \psi_n \rightarrow \varphi) \leftrightarrow (\psi_1 \rightarrow (\psi_2 \rightarrow \dots \rightarrow (\psi_n \rightarrow \varphi)))$$

is a tautology.

**Proposition 4.37**

K is closed under propositional deduction: if φ is deducible in propositional logic from assumptions ψ_1, \dots, ψ_n , then

$\vdash_K \psi_1, \dots, \vdash_K \psi_n$ implies $\vdash_K \varphi$.

Proof:

(1)	$\vdash_K \psi_1$	hypothesis
	⋮	
(n)	$\vdash_K \psi_n$	hypothesis
(n+1)	$\vdash_K \psi_1 \rightarrow (\psi_2 \rightarrow \dots \rightarrow (\psi_n \rightarrow \varphi))$	(Taut)
(n+2)	$\vdash_K \psi_2 \rightarrow \dots \rightarrow (\psi_{n-1} \rightarrow (\psi_n \rightarrow \varphi))$	(MP): (1), (n+1)
	⋮	
(2n-1)	$\vdash_K \psi_{n-1} \rightarrow (\psi_n \rightarrow \varphi)$	(MP): (n-2), (2n-2)
(2n)	$\vdash_K \psi_n \rightarrow \varphi$	(MP): (n-1), (2n-1)
(2n+1)	$\vdash_K \varphi$	(MP): (n), (2n) □

**Example 4.38**

$\vdash_K \varphi \rightarrow \psi$ implies $\vdash_K \Box \varphi \rightarrow \Box \psi$.

Proof: We give the following **K**-proof:

- | | | |
|-----|--|-----------------|
| (1) | $\vdash_K \varphi \rightarrow \psi$ | hypothesis |
| (2) | $\vdash_K \Box(\varphi \rightarrow \psi)$ | (GEN): (1) |
| (3) | $\vdash_K \Box(\varphi \rightarrow \psi) \rightarrow (\Box \varphi \rightarrow \Box \psi)$ | (K) |
| (4) | $\vdash_K \Box \varphi \rightarrow \Box \psi$ | (MP): (2), (3). |

**Example 4.39**

$\vdash_K \varphi \rightarrow \psi$ implies $\vdash_K \Diamond \varphi \rightarrow \Diamond \psi$.

Proof: We give the following **K**-proof:

- | | | |
|-----|--|--------------------------|
| (1) | $\vdash_K \varphi \rightarrow \psi$ | hypothesis |
| (2) | $\vdash_K (\varphi \rightarrow \psi) \rightarrow (\neg \psi \rightarrow \neg \varphi)$ | (Taut) |
| (3) | $\vdash_K \neg \psi \rightarrow \neg \varphi$ | (MP): (1), (2) |
| (4) | $\vdash_K \Box \neg \psi \rightarrow \Box \neg \varphi$ | Example 4.38: (3) |
| (5) | $\vdash_K (\Box \neg \psi \rightarrow \Box \neg \varphi) \rightarrow (\neg \Box \neg \varphi \rightarrow \neg \Box \neg \psi)$ | (Taut) |
| (6) | $\vdash_K \neg \Box \neg \varphi \rightarrow \neg \Box \neg \psi$ | (MP): (4), (5) |
| (7) | $\vdash_K \Diamond \varphi \rightarrow \Diamond \psi$ | definition of \Diamond |



K

Example 4.40

$$\vdash_K \Box(\varphi \wedge \psi) \rightarrow \Box\varphi \wedge \Box\psi.$$

Proof: We give the following **K**-proof:

- (1) $\vdash_K \varphi \wedge \psi \rightarrow \varphi$ (Taut)
- (2) $\vdash_K \Box(\varphi \wedge \psi) \rightarrow \Box\varphi$ Example 4.38: (1)
- (3) $\vdash_K \varphi \wedge \psi \rightarrow \psi$ (Taut)
- (4) $\vdash_K \Box(\varphi \wedge \psi) \rightarrow \Box\psi$ Example 4.38: (3)
- (5) $\vdash_K \Box(\varphi \wedge \psi) \rightarrow \Box\varphi \wedge \Box\psi$ Proposition 4.37, (2) and (4)

(5) is obtained as an application of Proposition 4.37 with

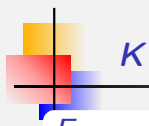
$$\psi_1 = \sigma_1 \rightarrow \sigma_2, \quad \psi_2 = \sigma_1 \rightarrow \sigma_3, \quad \varphi = \sigma_1 \rightarrow \sigma_2 \wedge \sigma_3,$$

where

$$\sigma_1 = \Box(\varphi \wedge \psi), \quad \sigma_2 = \Box\varphi, \quad \sigma_3 = \Box\psi.$$

□

201



K

Example 4.41

$$\vdash_K \Box\varphi \wedge \Box\psi \rightarrow \Box(\varphi \wedge \psi).$$

Proof: We give the following **K**-proof:

- (1) $\vdash_K \varphi \rightarrow (\psi \rightarrow (\varphi \wedge \psi))$ (Taut)
- (2) $\vdash_K \Box\varphi \rightarrow \Box(\psi \rightarrow (\varphi \wedge \psi))$ Ex. 4.38: (1)
- (3) $\vdash_K \Box(\psi \rightarrow (\varphi \wedge \psi)) \rightarrow (\Box\psi \rightarrow \Box(\varphi \wedge \psi))$ (K)
- (4) $\vdash_K \Box\varphi \rightarrow (\Box\psi \rightarrow \Box(\varphi \wedge \psi))$ Prop. 4.37, (2), (3)
- (5) $\vdash_K \Box\varphi \wedge \Box\psi \rightarrow \Box(\varphi \wedge \psi)$ Prop. 4.37, (4)

(4) is obtained as an application of Proposition 4.37 with

$$\psi_1 = \sigma_1 \rightarrow \sigma_2, \quad \psi_2 = \sigma_2 \rightarrow \sigma_3, \quad \varphi = \sigma_1 \rightarrow \sigma_3,$$

where

$$\sigma_1 = \Box\varphi, \quad \sigma_2 = \Box(\psi \rightarrow (\varphi \wedge \psi)), \quad \sigma_3 = \Box\psi \rightarrow \Box(\varphi \wedge \psi).$$

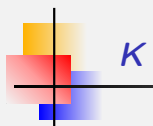
(5) is obtained as an application of Proposition 4.37 with

$$\psi_1 = \sigma_1 \rightarrow (\sigma_2 \rightarrow \sigma_3), \quad \varphi = (\sigma_1 \wedge \sigma_2) \rightarrow \sigma_3,$$

where $\sigma_1 = \Box\varphi, \quad \sigma_2 = \Box\psi, \quad \sigma_3 = \Box(\varphi \wedge \psi).$

□

202



K

Example 4.42

$$\vdash_K \Box\varphi \wedge \Box\psi \leftrightarrow \Box(\varphi \wedge \psi).$$

Proof: We give the following **K**-proof:

- (1) $\vdash_K \Box\varphi \wedge \Box\psi \rightarrow \Box(\varphi \wedge \psi)$ Example 4.41
- (2) $\vdash_K \Box(\varphi \wedge \psi) \rightarrow \Box\varphi \wedge \Box\psi$ Example 4.40
- (3) $\vdash_K \Box\varphi \wedge \Box\psi \leftrightarrow \Box(\varphi \wedge \psi)$

(3) is obtained as an application of Proposition 4.37 with

$$\psi_1 = \Box\varphi \wedge \Box\psi \rightarrow \Box(\varphi \wedge \psi), \quad \psi_2 = \Box(\varphi \wedge \psi) \rightarrow \Box\varphi \wedge \Box\psi$$

and

$$\varphi = \Box\varphi \wedge \Box\psi \leftrightarrow \Box(\varphi \wedge \psi).$$

□

203



K

The logic **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. **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 **K** with additional axioms.

204

By Lemma 4.30, for any set Γ of formulas, there exists the smallest normal modal logic that contains Γ .

Definition 4.43

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

Definition 4.44

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:

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

Definition 4.45

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

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

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

Theorem 4.46

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

Let Λ be a normal modal logic.

Definition 4.47

If $\varphi \in \Lambda$, we also say that φ is a **Λ -theorem** or a **theorem of Λ** and write $\vdash_{\Lambda} \varphi$. If $\varphi \notin \Lambda$, we write $\nvdash_{\Lambda} \varphi$.

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

For any formulas φ, ψ , the following hold:

- (i) If φ 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 $\vdash_{\Lambda} \Box\varphi$.

Remark 4.48

- ▶ $\vdash_K \varphi$ implies $\vdash_{\Lambda} \varphi$.
- ▶ If $\Gamma \subseteq \Lambda$, then $\vdash_{K\Gamma} \varphi$ implies $\vdash_{\Lambda} \varphi$.

Proposition 4.49

Λ is closed under propositional deduction: if φ is deducible in propositional logic from assumptions ψ_1, \dots, ψ_n , then

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

Proof: Exercise.

Definition 4.50

Let $\Gamma \cup \{\varphi\}$ be a set of formulas. We say that φ is **deducible in Λ from Γ** or that φ is **Λ -deducible from Γ** 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 φ is not Λ -deducible from Γ .

Remark 4.51

The following are equivalent:

- (i) $\Gamma \vdash_{\Lambda} \varphi$.
- (ii) There exist formulas $\psi_1, \dots, \psi_n \in \Gamma$ ($n \geq 0$) such that

$$\vdash_{\Lambda} \psi_1 \rightarrow (\psi_2 \rightarrow \dots \rightarrow (\psi_n \rightarrow \varphi)).$$

Proposition 4.52 (Basic properties)

Let φ be a formula and Γ, Δ 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 $\Delta \vdash_{\Lambda} \varphi$.

Proof: Exercise.

Let φ, ψ be formulas and Γ be a set of formulas,

Proposition 4.53

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

Proposition 4.54

- (i) If $\Gamma \vdash_{\Lambda} \varphi$ and ψ is deducible in propositional logic from φ , 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 4.55 (Deduction Theorem)

For any set of formulas Γ and any formulas φ, ψ ,

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

Definition 4.56

A set Γ of formulas is **Λ -consistent** if $\Gamma \not\vdash_{\Lambda} \varphi$ for any formula φ . If Γ is not Λ -consistent, we say that Γ is **Λ -inconsistent**. A formula φ is **Λ -consistent** if $\{\varphi\}$ is; otherwise, it is called **Λ -inconsistent**.

Proposition 4.57

Let Γ be a set of formulas. The following are equivalent:

- (i) Γ is Λ -inconsistent.
- (ii) There exists a formula ψ such that $\Gamma \vdash_{\Lambda} \psi$ and $\Gamma \vdash_{\Lambda} \neg\psi$.
- (iii) $\Gamma \vdash_{\Lambda} \perp$.

Proposition 4.58

Γ is Λ -consistent iff any finite subset of Γ is Λ -consistent.

Normal logics - soundness

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 4.59

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

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

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

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

213

Normal logics - soundness

Theorem 4.60 (Soundness theorem for \mathbf{K})

\mathbf{K} is **sound** with respect to the class of all frames.

Proof: We apply Theorem 4.29 and the fact that \mathbf{K} is the least normal logic. \square

214

Normal logics - completeness

Definition 4.61

A normal logic Λ 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 φ ,

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

Obviously, weak completeness is a particular case of strong completeness; just take $\Gamma = \emptyset$ in Definition 4.61.(i).

215

Normal logics - completeness

Remark 4.62

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

If a normal logic Λ 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 Γ such that $\Lambda_{\mathbf{S}}$ is the logic generated by Γ ; we say that Γ **axiomatizes \mathbf{S}** .

216

Theorem 4.63

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

217

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 4.64

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

218

Let

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

We use the notation T for the normal logic generated by (T).

Definition 4.65

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

Theorem 4.66

T is sound and strongly complete with respect to the class of reflexive frames.

219

Let

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

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

Definition 4.67

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

Theorem 4.68

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

220

Logic KD

Let

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

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

One can prove that $\vdash_{\mathbf{K}} (D) \leftrightarrow (D')$ (exercise).

Let **KD** be the normal logic generated by (D) (or, equivalently, by (D')).

Definition 4.69

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 4.70

KD is sound and strongly complete with respect to the class of serial frames.

221

Logic K5

Let

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

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

One can prove that $\vdash_{\mathbf{K}} (5) \leftrightarrow (5')$ (exercise).

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

Definition 4.71

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 4.72

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

222

Logic S4

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

Theorem 4.73

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

223

Logic S5

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

Proposition 4.74

$$\mathbf{S5} = \mathbf{KDB4} = \mathbf{KDB5} = \mathbf{KT5}.$$

Theorem 4.75

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

224

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 , 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:

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

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

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

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

225

- ▶ 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.

226

Definition 4.76

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

- ▶ Λ contains all propositional tautologies and is closed under modus ponens.
- ▶ Λ contains all formulas

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

where φ, ψ are formulas and $i \in I$.

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

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

227

- ▶ 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 Γ is also denoted by $\mathbf{K}\Gamma$. Furthermore, $\mathbf{K}\Gamma = \{\varphi \mid \vdash_{\mathbf{K}\Gamma} \varphi\}$.
- ▶ The definitions of Λ -deducibility, Λ -consistence, soundness and weak and strong completeness are unchanged.

228

Epistemic Logics

Textbook:

R. Fagin, J.Y. Halpern, Y. Moses, M. Vardi, [Reasoning About Knowledge](#), MIT Press, 2004

229

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.

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

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

230

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) φ** .
- ▶ 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) φ** .

231

Epistemic logics

Definition 5.1

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

- ▶ Λ contains all propositional tautologies and is closed under modus ponens.

- ▶ Λ contains all formulas

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

where φ, ψ are formulas and $i \in Ag$.

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

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

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

232

Recall the following axioms:

$$\begin{aligned} (T) \quad & K_i \varphi \rightarrow \varphi \\ (D') \quad & \neg K_i (\varphi \wedge \neg \varphi) \\ (B) \quad & \varphi \rightarrow K_i \neg K_i \neg \varphi \end{aligned}$$

Properties of knowledge

- ▶ Axiom (T) is called the **verity** or **knowledge** axiom: If an agent knows φ , then φ 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 φ and $\neg\varphi$. **An agent cannot know a contradiction.**
- ▶ Axiom (B) says that if φ holds, then an agent knows that it does not know $\neg\varphi$.

233

Recall the following axioms:

$$\begin{aligned} (4) \quad & K_i \varphi \rightarrow K_i K_i \varphi \\ (5') \quad & \neg K_i \varphi \rightarrow K_i \neg K_i \varphi \end{aligned}$$

Properties of knowledge

- ▶ Axiom (4) is **positive introspection**: if an agent knows φ , it knows that it knows φ . **An agent knows what it knows.**
- ▶ Axiom (5') is **negative introspection**: if an agent does not know φ , it knows that it does not know φ . **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.

234

Let $S5 = KD'B4 = KD'B5' = KT5'$. S5 is considered as the logic of **idealised knowledge**.

Theorem 5.2

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

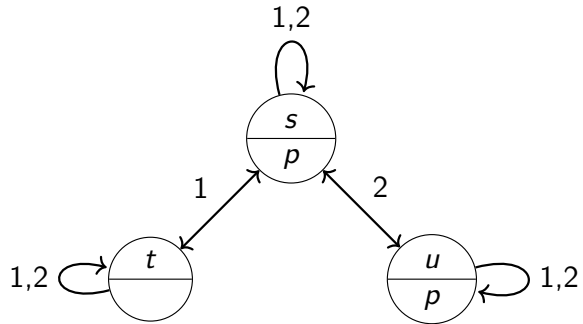
235

A model $M = (W, \mathcal{K}_1, \dots, \mathcal{K}_n, V)$ is represented as a labeled directed graph:

- ▶ the nodes of the graph are the states of the model;
- ▶ the label of each node $w \in W$ describes which atomic propositions are true at state w ;
- ▶ we label edges by sets of agents; the label of the edge from node w to node v includes i iff $\mathcal{K}_i wv$ holds.

236

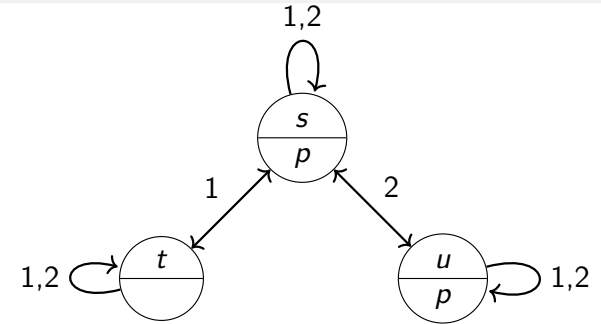
Example



We have that $Ag = \{1, 2\}$, $PROP = \{p\}$ and $\mathcal{M} = (W, \mathcal{K}_1, \mathcal{K}_2, V)$, where

- ▶ $W = \{s, t, u\}$.
- ▶ $\mathcal{K}_1 = \{(s, s), (t, t), (u, u), (s, t), (t, s)\}$.
- ▶ $\mathcal{K}_2 = \{(s, s), (t, t), (u, u), (s, u), (u, s)\}$.
- ▶ $V(p) = \{s, u\}$.

Example

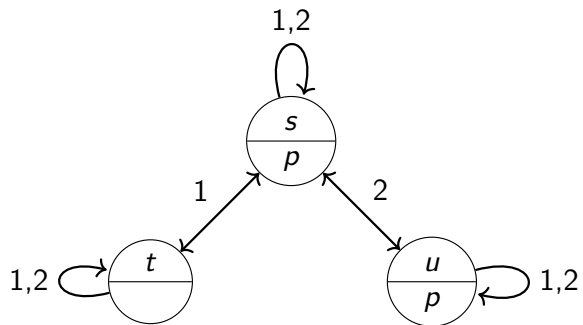


- ▶ $\mathcal{M}, s \Vdash p \wedge \neg K_1 p$.

Proof: We have that $s \in V(p)$, hence $\mathcal{M}, s \Vdash p$. Since $\mathcal{K}_1 st$ and $\mathcal{M}, t \not\Vdash p$, it follows that $\mathcal{M}, s \not\Vdash K_1 p$, hence $\mathcal{M}, s \Vdash \neg K_1 p$. Thus, $\mathcal{M}, s \Vdash p \wedge \neg K_1 p$. □

In state s , p is true, but agent 1 does not know it, since in state s it considers both s and t possible. We say that agent 1 cannot distinguish s from t . Agent 1's information is insufficient to enable it to distinguish whether the actual state is s or t .

Example

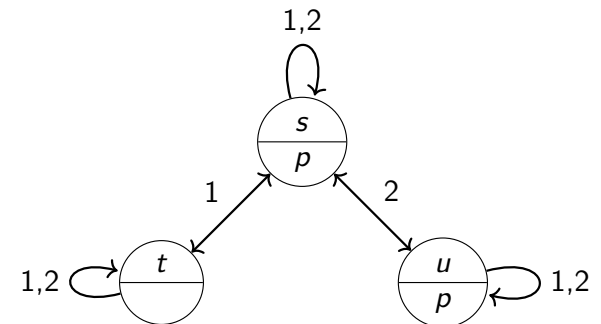


- ▶ $\mathcal{M}, s \Vdash K_2 p$.

Proof: We have that $\mathcal{M}, s \Vdash K_2 p$ iff for all $v \in W$, $\mathcal{K}_2 sv$ implies $\mathcal{M}, v \Vdash p$ iff $\mathcal{M}, s \Vdash p$ and $\mathcal{M}, u \Vdash p$ (as $\mathcal{K}_2 ss, \mathcal{K}_2 su$, but we don't have that $\mathcal{K}_2 st$), which is true. □

In state s , agent 2 knows that p is true, since p is true in both states that agent 2 considers possible at s (namely s and u).

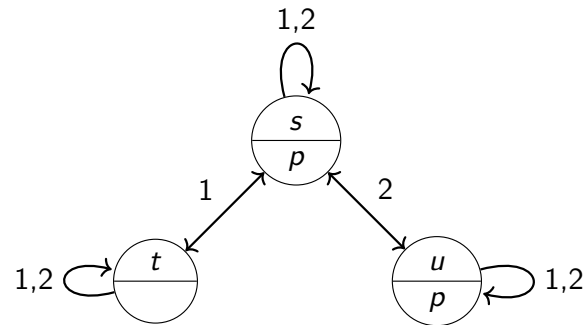
Example



- ▶ $\mathcal{M}, s \Vdash \neg K_2 \neg K_1 p$.

Proof: We have that $\mathcal{M}, s \Vdash \neg K_2 \neg K_1 p$ iff $\mathcal{M}, s \not\Vdash K_2 \neg K_1 p$ iff there exists $v \in W$ such that $\mathcal{K}_2 sv$ and $\mathcal{M}, v \not\Vdash \neg K_1 p$ iff there exists $v \in W$ such that $\mathcal{K}_2 sv$ and $\mathcal{M}, v \Vdash K_1 p$. Take $v := u$. Then $\mathcal{K}_2 su$ and $\mathcal{M}, u \Vdash K_1 p$, since $\mathcal{M}, u \Vdash p$ and $\mathcal{K}_1 uw$ iff $w = u$. □

Example



- ▶ $\mathcal{M}, s \models \neg K_2 \neg K_1 p$.

Although agent 2 knows that p is true in s , it does not know that agent 1 does not know this fact. Why? Because in a state that agent 2 considers possible, namely u , agent 1 does know that p is true, while in another state agent 2 considers possible, namely s , agent 1 does not know this fact.

$$Ag = \{1, 2\}$$

- ▶ Suppose that we have a deck consisting of three cards labeled A , B , and C . Agents 1 and 2 each get one of these cards; the third card is left face down.
- ▶ A possible world is characterized by describing the cards held by each agent. For example, in the world (A, B) , agent 1 holds card A and agent 2 holds card B (while card C is face down).
- ▶ The set of possible worlds is

$$W = \{(A, B), (A, C), (B, A), (B, C), (C, A), (C, B)\}.$$
- ▶ In a world such as (A, B) , agent 1 thinks two worlds are possible: (A, B) and (A, C) . Agent 1 knows that he has card A , but considers it possible that agent 2 could hold either card B or card C .

- ▶ Similarly, in world (A, B) , agent 2 also considers two worlds are possible: (A, B) and (C, B) .
- ▶ In general, in a world (X, Y) , agent 1 considers (X, Y) and (X, Z) possible, while agent 2 considers (X, Y) and (Z, Y) possible, where Z is different from both X and Y .

- ▶ We can easily construct the \mathcal{K}_1 and \mathcal{K}_2 relations.
- ▶ It is easy to check that they are indeed equivalence relations.
- ▶ This is because an agent's possibility relation is determined by the information he has, namely, the card he is holding.

We describe the frame $\mathcal{F}_c = (W, \mathcal{K}_1, \mathcal{K}_2)$ for the card game as a labeled graph. Since the relations are equivalence relations, we omit the self loops and the arrows on edges for simplicity (if there is an edge from state w to state v , there must be an edge from v to w as well, by symmetry).

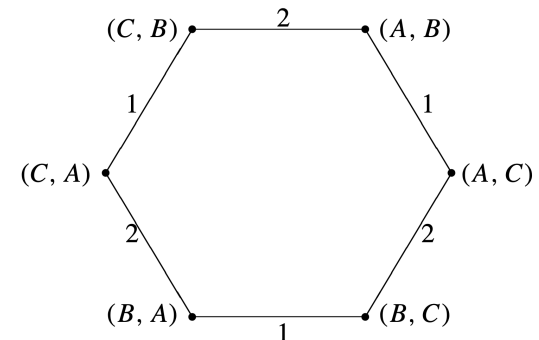
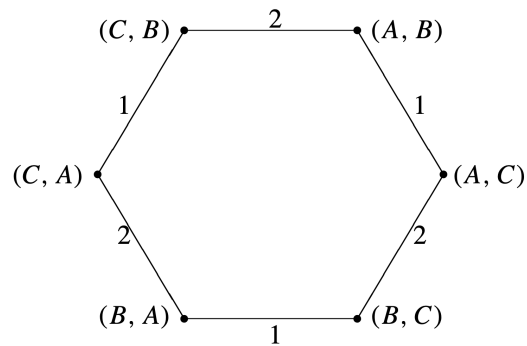


Figure 7: Frame describing a simple card game

A simple card game



- ▶ In the world (A, B) , agent 1 knows that the world (B, C) cannot be the case. This is captured by the fact that there is no edge from (A, B) to (B, C) labeled 1.
- ▶ Nevertheless, agent 1 considers it possible that agent 2 considers it possible that (B, C) is the case. This is captured by the fact that there is an edge labeled 1 from (A, B) to (A, C) , from which there is an edge labeled 2 to (B, C) .

245

A simple card game

We still have not defined the model to be used in this example.

Define the set PROP of atomic propositions as

$$PROP = \{iX \mid i \in \{1, 2\}, X \in \{A, B, C\}\}.$$

iX will be interpreted as **agent i holds card X** . Given this interpretation, we define the valuation V in the obvious way:

$$V(iX) = \begin{cases} \{(X, Z) \mid Z \in \{A, B, C\} \setminus \{X\}\} & \text{if } i = 1 \\ \{(Z, X) \mid Z \in \{A, B, C\} \setminus \{X\}\} & \text{if } i = 2. \end{cases}$$

246

A simple card game

Let $\mathcal{M}_c = (\mathcal{F}_c, V)$ be the model describing this card game.

Examples

- ▶ $\mathcal{M}_c, (A, B) \models 1A \wedge 2B$.
- ▶ $\mathcal{M}_c, (A, B) \models K_1(2B \vee 2C)$: agent 1 knows that agent 2 holds either B or C .
- ▶ $\mathcal{M}_c, (A, B) \models K_1 \neg 2A$: agent 1 knows that agent 2 does not hold an A .
- ▶ $\mathcal{M}_c, (A, B) \models K_1 \neg K_2 1A$: agent 1 knows that agent 2 does not know that agent 1 holds an A .

247

Common and distributed knowledge

We need to reason about **knowledge in a group** and using this understanding to help us analyze multiagent systems.

- ▶ An agent in a group must take into account not only facts that are true about the world, but also the knowledge of other agents in the group.
- ▶ For example, in a bargaining situation, the seller of a car must consider what the potential buyer knows about the car's value. The buyer must also consider what the seller knows about what the buyer knows about the car's value, and so on.
- ▶ Such reasoning can get rather convoluted. Example: "Dean doesn't know whether Nixon knows that Dean knows that Nixon knows that McCord burgled O'Brien's office at Watergate."
- ▶ But this is precisely the type of reasoning that is needed when analyzing the knowledge of agents in a group.

248

Common and distributed knowledge

We are often interested in situations in which **everyone** in the group knows a fact.

Example

A society certainly wants all drivers to know that a **red light** means **stop** and a **green light** means **go**. Suppose we assume that every driver in the society knows this fact and follows the rules. A driver does not feel safe, unless she also knows that everyone else knows and is following the rules. Otherwise, a driver may consider it possible that, although she knows the rules, some other driver does not, and that driver may run a red light.

249

Common and distributed knowledge

- ▶ In some cases we also need to consider the state in which simultaneously everyone knows a fact φ , everyone knows that everyone knows φ , everyone knows that everyone knows that everyone knows φ , and so on. We say that the group has **common knowledge** of φ .
- ▶ The notion of common knowledge was first studied by the philosopher **David Lewis** in the context of **conventions**: in order for something to be a convention, it must be common knowledge among the members of a group.
- ▶ John McCarthy, in the context of studying common-sense reasoning, characterized common knowledge as **what any fool knows**.

Example

The convention that **green** means **go** and **red** means **stop** is presumably common knowledge among the drivers in our society.

250

Common and distributed knowledge

- ▶ Common knowledge also arises in **discourse understanding**.
- ▶ Suppose that Ann asks Bob "What did you think of the movie?" referring to the movie Star Wars they have just seen. Ann and Bob must both know that **the movie** refers to **Star Wars**, Ann must know that Bob knows (so that she can be sure that Bob will give a reasonable answer to her question), Bob must know that Ann knows that Bob knows (so that Bob knows that Ann will respond appropriately to his answer), and so on.
- ▶ There must be common knowledge of what movie is meant in order for Bob to answer the question appropriately.
- ▶ Common knowledge also turns out to be a prerequisite for achieving agreement.
- ▶ That is why common knowledge is a crucial notion in the analysis of interacting groups of agents.

251

Common and distributed knowledge

A group has **distributed knowledge** of a fact φ if the knowledge of φ is distributed among its members, so that by using their knowledge together the members of the group can deduce φ , even though it may be the case that no member of the group individually knows φ .

Example

Assume that Alice knows that Bob is in love with either Carol or Susan, and Charlie knows that Bob is not in love with Carol. Then together Alice and Charlie have distributed knowledge of the fact that Bob is in love with Susan, although neither Alice nor Charlie individually has this knowledge.

252

Let $\emptyset \neq G \subseteq Ag$ be a group of agents.

- ▶ Define, for every φ ,

$$E_G\varphi = \bigwedge_{i \in G} K_i\varphi.$$

- ▶ $E_G\varphi$ is read as **everyone in the group G knows φ** .
- ▶ For every model \mathcal{M} and every $w \in \mathcal{M}$,
 $\mathcal{M}, w \models E_G\varphi$ iff $\mathcal{M}, w \models K_i\varphi$ for all $i \in G$.

The language ML_{Ag} does not allow us to express the notions of common knowledge and distributed knowledge.

Let ML_{Ag}^{CD} be the language obtained by adding to ML_{Ag} the following modal operators for any $\emptyset \neq G \subseteq Ag$:

- ▶ C_G , read as **it is common knowledge among the agents in G** ;
- ▶ D_G , read as **it is distributed knowledge among the agents in G** .

Formulas of ML_{Ag}^{CD} are defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid K_i\varphi \mid C_G\varphi \mid D_G\varphi,$$

where $p \in PROP$, $i \in Ag$ and $\emptyset \neq G \subseteq Ag$.

We omit the subscript G when G is the set of all agents.

Let $\emptyset \neq G \subseteq Ag$ be a group of agents.

We define $E_G^k\varphi$ ($k \geq 0$) inductively:

$$E_G^0\varphi = \varphi, \quad E_G^{k+1}\varphi = E_G E_G^k\varphi.$$

Let \mathcal{M} be a model and $w \in \mathcal{M}$. We extend the definition of the satisfaction relation with the following clause:

$$\mathcal{M}, w \models C_G\varphi \text{ iff } \mathcal{M}, w \models E_G^k\varphi \text{ for all } k = 1, 2, \dots$$

Thus, the formula $C_G\varphi$ is true iff everyone in G knows φ , everyone in G knows that everyone in G knows φ , etc.

Our definition of common knowledge has a graph-theoretical interpretation.

Let \mathcal{M} be a model.

Definition 5.3

Let w, v be states in \mathcal{M} .

- ▶ We say that v is **G -reachable from w in k steps** ($k \geq 1$) if there exist states $u_0, u_1, \dots, u_k \in \mathcal{M}$ such that $u_0 = w$, $u_k = v$ and for all $j = 0, \dots, k-1$, there exists $i \in G$ such that $\mathcal{K}_i u_j u_{j+1}$.
- ▶ v is **G -reachable from w** if v is G -reachable from w in k steps for some $k \geq 1$.

Thus, v is G -reachable from w iff there is a path in the graph from w to v whose edges are labeled by members of G .

Proposition 5.4

Let w be a state in \mathcal{M} .

- ▶ The following are equivalent for every $k \geq 1$:
 - ▶ $\mathcal{M}, w \Vdash E_G^k \varphi$;
 - ▶ $\mathcal{M}, v \Vdash \varphi$ for all states v that are G -reachable from w in k steps.
- ▶ $\mathcal{M}, w \Vdash C_G \varphi$ iff $\mathcal{M}, v \Vdash \varphi$ for all states v that are G -reachable from w .

A group G has **distributed knowledge** of φ if the combined knowledge of the members of G implies φ .

- ▶ The question is how can we capture the idea of combining knowledge in our framework.
- ▶ The answer is: we combine the knowledge of the agents in group G by eliminating all worlds that some agent in G considers impossible.

Let \mathcal{M} be a model and $w \in \mathcal{M}$. We extend the definition of the satisfaction relation with the following clause:

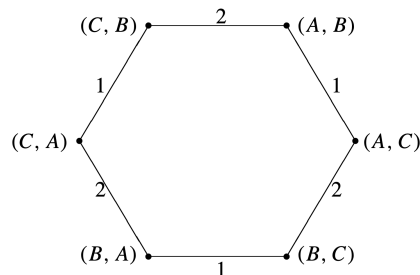
$$\begin{aligned} \mathcal{M}, w \Vdash D_G \varphi & \text{ iff } \mathcal{M}, v \Vdash \varphi \text{ for all } v \text{ such that } (w, v) \in \bigcap_{i \in G} \mathcal{K}_i \\ & \text{ iff } \mathcal{M}, v \Vdash \varphi \text{ for all } v \text{ such that } \mathcal{K}_i wv \text{ for all } i \in G. \end{aligned}$$

Example - the card game

Let $\mathcal{M}_c = (\mathcal{F}_c, V)$ be the model describing the simple card game.

- ▶ $PROP = \{iX \mid i \in \{1, 2\}, X \in \{A, B, C\}\}$.
- ▶ iX read as **agent i holds card X**
- ▶ $V(iX) = \begin{cases} \{(X, Z) \mid Z \in \{A, B, C\} \setminus \{X\}\} & \text{if } i = 1 \\ \{(Z, X) \mid Z \in \{A, B, C\} \setminus \{X\}\} & \text{if } i = 2. \end{cases}$

\mathcal{F}_c is given by



Example - the card game

Let $G = \{1, 2\}$.

- ▶ $\mathcal{M}_c \Vdash C_G(1A \vee 1B \vee 1C)$: it is common knowledge that agent 1 holds one of the cards A , B , and C .
- ▶ $\mathcal{M}_c \Vdash C_G(1B \rightarrow (2A \vee 2C))$: it is common knowledge that if agent 1 holds card B , then agent 2 holds either card A or card C .
- ▶ $\mathcal{M}_c, (A, B) \Vdash D_G(1A \wedge 2B)$: if the agents could combine their knowledge, they would know that in world (A, B) , agent 1 holds card A and agent 2 holds card B .

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.

261

Muddy children puzzle

For simplicity, let us call a child

- ▶ **muddy** if he has a muddy forehead;
- ▶ **clean** if he does not have a muddy forehead.

$k = 1$

- ▶ 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.

262

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.

263

Muddy children puzzle

We shall analyse the muddy children puzzle using epistemic logic.

We assume that it is common knowledge that

- ▶ the father is truthful,
- ▶ all the children can and do hear the father,
- ▶ all the children can and do see which of the other children besides themselves have muddy foreheads,
- ▶ none of the children can see his own forehead,
- ▶ all the children are truthful and (extremely) intelligent.

264

Muddy children puzzle

Suppose that there are n children; we number them $1, \dots, n$. Thus, we take $Ag = \{1, \dots, n\}$.

- ▶ First consider the situation before the father speaks.
- ▶ Some of the children are muddy, while the rest are clean.
- ▶ We describe a possible situation by an n -tuple of 0's and 1's of the form (x_1, \dots, x_n) , where $x_i = 1$ if child i is muddy, and $x_i = 0$ otherwise.
- ▶ There are 2^n possible situations.

265

Muddy children puzzle

$n = 3$

- ▶ Suppose that the actual situation is described by the tuple $(1, 0, 1)$, that says that child 1 and child 3 are muddy, while child 2 is clean.
- ▶ What situations does child 1 consider possible before the father speaks?
- ▶ Since child 1 can see the foreheads of all the children besides himself, his only doubt is about whether he is muddy or clean. Thus child 1 considers two situations possible: $(1, 0, 1)$ (the actual situation) and $(0, 0, 1)$. Similarly, child 2 considers two situations possible: $(1, 0, 1)$ and $(1, 1, 1)$.

In general, child i has the same information in two possible situations exactly if they agree in all components except possibly the i th component.

266

Muddy children puzzle

We can capture the general situation by the frame

$$\mathcal{F} = (W, \mathcal{K}_1, \dots, \mathcal{K}_n),$$

where

- ▶ $W = \{(x_1, \dots, x_n) \mid x_i \in \{0, 1\} \text{ for all } i = 1, \dots, n\}$. Thus, W has 2^n states.
- ▶ For every $i = 1, \dots, n$,
 $\mathcal{K}_i wv$ iff w and v agree in all components except possibly the i th component.
- ▶ One can easily see that \mathcal{K}_i 's are equivalence relations.

Thus, \mathcal{F} is a frame for the epistemic logic **S5**.

267

Muddy children puzzle

It remains to define $PROP$ and the valuation $V : PROP \rightarrow 2^W$.

- ▶ Since we want to reason about whether or not a given child is muddy, we take $PROP = \{p_1, \dots, p_n, p\}$, where, intuitively, p_i stands for **child i is muddy**, while p stands for **at least one child is muddy**.
- ▶ We define V as follows:
 $V(p_i) = \{(x_1, \dots, x_n) \in W \mid x_i = 1\}$,
 $V(p) = \{(x_1, \dots, x_n) \in W \mid x_j = 1 \text{ for some } j = 1, \dots, n\}$.
- ▶ It follows that
 $\mathcal{M}, (x_1, \dots, x_n) \models p_i$ iff $x_i = 1$,
 $\mathcal{M}, (x_1, \dots, x_n) \models p$ iff $x_j = 1$ for some $j = 1, \dots, n$.

We have a model with 2^n nodes, each described by an n -tuple of 0's and 1's, such that two nodes are joined by an edge exactly if they differ in at most one component.

268

Muddy children puzzle

Recall that we omit self-loops and the arrows on edges.

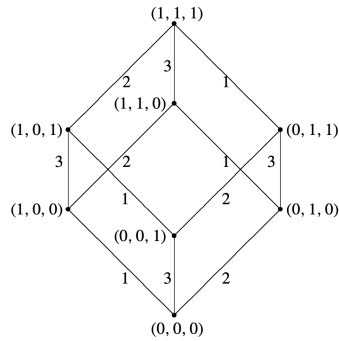
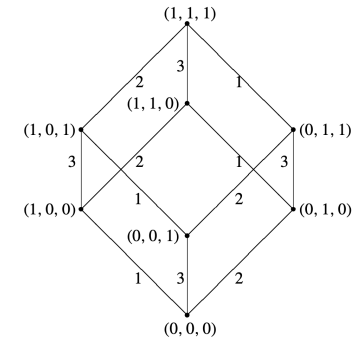


Figure 8: Frame for the muddy children puzzle with $n = 3$

- ▶ $\mathcal{M}, (1, 0, 1) \models K_1 \neg p_2$;
- ▶ $\mathcal{M}, (1, 0, 1) \models K_1 p_3$;
- ▶ $\mathcal{M}, (1, 0, 1) \models \neg K_1 p_1$;

269

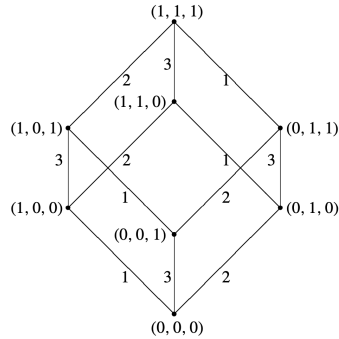
Muddy children puzzle



- ▶ $\mathcal{M} \models C(p_2 \rightarrow K_1 p_2)$: it is common knowledge that if child 2 is muddy, then child 1 knows it.
- ▶ $\mathcal{M} \models C(\neg p_2 \rightarrow K_1 \neg p_2)$: it is common knowledge that if child 2 is clean, then child 1 knows it.

270

Muddy children puzzle



- ▶ $\mathcal{M}, (1, 0, 1) \models E p$: in $(1, 0, 1)$, every child knows that at least one child is muddy even before the father speaks;
- ▶ $\mathcal{M}, (1, 0, 1) \models \neg E^2 p$: p is not true at the state $(0, 0, 0)$ that is reachable in two steps from $(1, 0, 1)$.

271

Muddy children puzzle

One can check that in the general case, if we have n children of whom k are muddy (so that the situation is described by an n -tuple exactly k of whose components are 1's), then $E^{k-1} p$ is true, but $E^k p$ is not, since each state reachable in $k - 1$ steps has at least one 1 (and so there is at least one muddy child), but the tuple $(0, \dots, 0)$ is reachable in k steps.

272

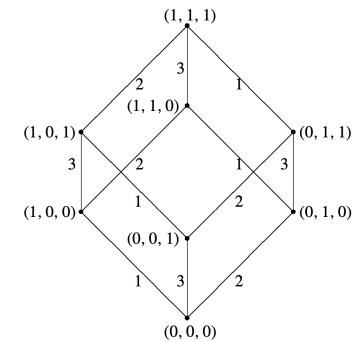
Muddy children puzzle

Let us consider what happens after the father speaks.

- ▶ The father says p , which is already known to all the children if there are two or more muddy children.
- ▶ Nevertheless, the state of knowledge changes, even if all the children already know p .

273

Muddy children puzzle



- ▶ In $(1, 0, 1)$, child 1 considers the situation $(0, 0, 1)$ possible and in $(0, 0, 1)$ child 3 considers $(0, 0, 0)$ possible.
- ▶ In $(1, 0, 1)$, before the father speaks, although everyone knows that at least one is muddy, child 1 thinks it possible that child 3 thinks it possible that none of the children is muddy.
- ▶ After the father speaks, it becomes common knowledge that at least one child is muddy.

274

Muddy children puzzle

- ▶ In the general case, we can represent the change in the group's state of knowledge graphically by simply removing the point $(0, 0, \dots, 0)$ from the cube.
- ▶ More accurately, what happens is that the node $(0, 0, \dots, 0)$ remains, but all the edges between $(0, 0, \dots, 0)$ and nodes with exactly one 1 disappear, since it is common knowledge that even if only one child is muddy, after the father speaks that child will not consider it possible that no one is muddy.

275

Muddy children puzzle

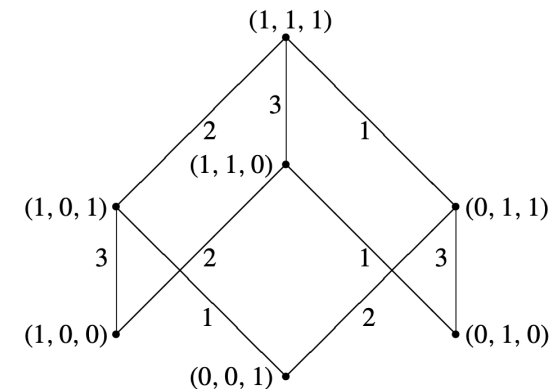


Figure 9: Frame for $n = 3$ after the father speaks

276



Muddy children puzzle

Let us show that each time the children respond to the father's question with a **No**, the group's state of knowledge changes and the cube is further truncated.

- ▶ Consider what happens after the children respond **No** to the father's first question.
- ▶ Now all the nodes with exactly one 1 can be eliminated. (More accurately, the edges to these nodes from nodes with exactly two 1's all disappear from the graph.)
- ▶ Nodes with one or fewer 1's are no longer reachable from nodes with two or more 1's.

277



Muddy children puzzle

- ▶ If the actual situation were described by, say, the tuple $(1, 0, \dots, 0)$, then child 1 would initially consider two situations possible: $(1, 0, \dots, 0)$, and $(0, 0, \dots, 0)$.
- ▶ Since once the father speaks it is common knowledge that $(0, 0, \dots, 0)$ is not possible, he would then know that the situation is described by $(1, 0, \dots, 0)$, and thus would know that he is muddy.
- ▶ Once everyone answers **No** to the father's first question, it is common knowledge that the situation cannot be $(1, 0, \dots, 0)$.
- ▶ Similar reasoning allows us to eliminate every situation with exactly one 1. Thus, after all the children have answered **No** to the father's first question, it is common knowledge that **at least two children are muddy**.

278



Muddy children puzzle

- ▶ Further arguments in the same spirit can be used to show that after the children answer **No** k times, we can eliminate all the nodes with at most k 1's (or, more accurately, disconnect these nodes from the rest of the graph).
- ▶ We thus have a sequence of frames, describing the children's knowledge at every step in the process.
- ▶ Essentially, what is going on is that if, in some node w , it becomes common knowledge that a node v is impossible, then for every node u reachable from w , the edge from u to v (if there is one) is eliminated.

279



Muddy children puzzle

- ▶ After k rounds of questioning, it is common knowledge that at least $k + 1$ children are muddy.
- ▶ If the true situation is described by a tuple with exactly $k + 1$ 1's, then before the father asks the question for the $(k + 1)$ st time, the muddy children will know the exact situation, and in particular will know they are muddy, and consequently will answer **Yes**.
- ▶ Note that they could not answer **Yes** any earlier, since up to this point each muddy child considers it possible that he or she is clean.

280

Muddy children puzzle

- ▶ According to the way we are modeling **knowledge** in this context, a child **knows** a fact if the fact follows from his or her current information.
- ▶ However, if one of the children were not particularly bright, then he might not be able to figure out that he **knew** that he is muddy, even though in principle he had enough information to do so.
- ▶ To answer **Yes** to the father's question, the child must actually be aware of the consequences of his information.
- ▶ Our definition implicitly assumes that (it is common knowledge that) all reasoners are **logically omniscient**, that is they are smart enough to compute all the consequences of the information that they have.
- ▶ Furthermore, this **logical omniscience** is **common knowledge**.

281

Muddy children puzzle

- ▶ Consider now the situation in which the father **does not initially say p** .
- ▶ Before the father speaks the situation is described by the n -dimensional cube.
- ▶ When the father asks for the first time **Does any of you know whether you have mud on your own forehead?**, clearly all the children say **No**, since in every situation each child considers possible a situation in which he is clean.
- ▶ No information is gained from this answer, so the situation still can be represented by the n -dimensional cube.
- ▶ One shows by induction on m that it is common knowledge that the father's m th question is also answered **No** and the state of knowledge after the father asks the m th question is still described by the cube.
- ▶ Hence, children's state of knowledge never changes, no matter how many times the father asks questions.

282

Partition model of knowledge

Partition models of knowledge are defined in
Yoav Shoham, Kevin Leyton-Brown, **Multiagents Systems**,
Cambridge University Press, 2009

Let $n \geq 1$ and $AG = \{1, \dots, n\}$ be the set of agents.

Definition 5.5 (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 .

283

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 .

284

- ▶ For each $i = 1, \dots, n$, let R_i be the corresponding equivalence relation.
- ▶ Denote by $I_i(w)$ the equivalence class of w in the relation R_i .
- ▶ If the actual world is w , then $I_i(w)$ is the set of possible worlds that agent i cannot distinguish from w .
- ▶ $\mathcal{F} = (W, R_1, \dots, R_n)$ is a frame for the epistemic logic **S5**.

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

285

Definition 5.6 (Partition model)

A **partition model** over a language Σ 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 φ is satisfied.

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

286

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

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

Definition 5.7 (Logical entailment for partition models)

Let $\mathcal{P}_M = (W, I_1, \dots, I_n, \pi)$ be a partition model over Σ , 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.

287

Multiagent systems

Textbook:

R. Fagin, J.Y. Halpern, Y. Moses, M. Vardi, [Reasoning About Knowledge](#), MIT Press, 2004

288



Runs and systems

289



Multiagent systems

- ▶ A **multiagent system** is any collection of interacting agents.
- ▶ One of the major application areas of **reasoning about knowledge**: multiagent systems.

Examples

- ▶ The children and the father in the muddy children puzzle are **agents** in a multiagent system.
- ▶ A game such as poker is a multiagent system. We refer to agents as **players**.
- ▶ A distributed system consisting of processes in a computer network running a particular protocol is a multiagent system. We refer to agents as **processes** or **sites**.

290



Multiagent systems

We define in the sequel a **formal model** of **multiagent systems** that is **general enough** to allow us to capture all the **important features** of multiagent systems.

Key assumptions

- ▶ We look at the system at **any point in time**, each of the agents is in a **unique state**. We refer to this as the agent's **local state**.
- ▶ An agent's local state encapsulates **all the information** to which the **agent has access**.
- ▶ We do not make any additional assumptions about the local state.

291



Multiagent systems

Assume that $n \geq 1$ and $AG = \{1, \dots, n\}$ is the set of agents.

- ▶ As each agent has a state, we think of the whole system as being in some state.
- ▶ First idea: take the system's state to be a tuple of the form (s_1, \dots, s_n) , where s_i is the local state of agent i .
- ▶ However, in general, **more than** just the local states of the agents may be relevant when analyzing a system.

Examples

- ▶ Consider a message-passing system where processes send messages back and forth along communication lines. We would like to know about messages that are in transit or about whether a communication line is up or down.
- ▶ Consider a system of sensors observing some terrain. We need to include features of the terrain in a description of the state of the system.

292

We conceptually divide a system into two components:

- ▶ the **agents** and
- ▶ the **environment**,

where we view the environment as **everything else that is relevant**. The environment can be viewed as just another agent, though it typically plays a special role in our analyses.

We define a **global state** of a system to be an $(n + 1)$ -tuple of the form (s_e, s_1, \dots, s_n) , where s_e is the state of the **environment** and s_i is the local state of **agent i** .

- ▶ A global state describes the system at a given point in time. But a system is not a static entity; it constantly changes.
- ▶ Since we are mainly interested in how systems change over time, we need to build time into our model.
- ▶ We define a **run** to be a **function from time to global states**.
- ▶ Intuitively, a run is a complete description of how the system's global state evolves over time.
- ▶ We take **time** to range over the **natural numbers**. Thus, time steps are discrete and time is infinite.
- ▶ The initial global state of the system in a possible execution r is $r(0)$, the next global state is $r(1)$, and so on.
- ▶ The assumption that time is discrete is a natural one, as computers proceed in discrete time steps. Allowing time to be infinite makes it easier to model situations where **there is no a priori time bound** on how long the system will run.

- ▶ We assume that time is measured on **some clock external** to the system.
- ▶ We **do not assume** that agents in the system necessarily **have access** to this clock; at time m measured on the external clock, agent i need not know it is time m .
- ▶ If an agent does know the time, then this information would be encoded in his local state.
- ▶ This external clock need not measure **real time**.
- ▶ We model the external clock in whatever way makes it easiest for us to analyze the system.
- ▶ In the case of the muddy children puzzle, there could be **one tick of the clock** for every round of questions by the father and every round of answers to the father's question.
- ▶ In a poker game, there could be **one tick of the clock** each time someone bets or discards.

A system can have **many possible runs**, since the system's global state can evolve in many possible ways: there are a number of possible initial states and many things that could happen from each initial global state.

- ▶ The **basic idea** is to define a **system** to be a **nonempty set of runs**.
- ▶ Instead of trying to **model the system directly**, the definition **models the possible behaviors of the system**.
- ▶ The set of runs is **nonempty**, as the system we are modeling should have some behaviors.

We use the term **system** in two ways: as the **real-life collection of interacting agents** or as **a set of runs**.

- ▶ Let L_e be a set of possible states for the **environment** and let L_i be a set of possible local states for **agent i** for $i = 1, \dots, n$.
- ▶ We take $\mathcal{G} = L_e \times L_1 \times \dots \times L_n$ to be the set of **global states**.

Definition 6.1

A **run over \mathcal{G}** is a function $r : \mathbb{N} \rightarrow \mathcal{G}$.

A run r over \mathcal{G} can be identified with the sequence $(r(m))_{m \in \mathbb{N}}$ of global states in \mathcal{G} .

Definition 6.2

We refer to a pair (r, m) consisting of a run r and time m as a **point**. We say that $r(m)$ is the **global state at the point (r, m)** .

A **round** takes place between two time points. We define round m in run r to take place between time $m - 1$ and time m . We view an agent as performing an **action** during a round.

Let (r, m) be a point and $r(m) = (s_e, s_1, \dots, s_n)$ be the global state at (r, m) . We define

$$r_e(m) = s_e \text{ and } r_i(m) = s_i \text{ for all } i = 1, \dots, n.$$

Thus, $r_i(m)$ is agent i 's local state at the point (r, m) .

Definition 6.3

A **system \mathcal{R} over \mathcal{G}** is a set of runs over \mathcal{G} . We say that (r, m) is a **point in system \mathcal{R}** if $r \in \mathcal{R}$. We denote by $\mathcal{P}_{\mathcal{R}}$ the set of points of \mathcal{R} .

In practice, the appropriate set of runs will be chosen by the system designer or the person analyzing the system.

- ▶ Imagine we have two processes, say a **sender S** and a **receiver R** , that communicate over a communication line.
- ▶ The sender starts with **one bit** (either 0 or 1) that it wants to communicate to the receiver.
- ▶ The communication line is faulty. There is no guarantee that a message sent by either S or R will be received.
- ▶ We assume that a message is either received in the same round that it is sent, or lost altogether. Thus, rounds are long enough for a message to be sent and delivered.
- ▶ This type of message loss is the only possible faulty behavior in the system.

- ▶ Because of the uncertainty regarding possible message loss, S sends the bit to R in every round, until S receives a message from R acknowledging receipt of the bit. We call this message from R an **ack** message.
- ▶ R starts **sending** the **ack** message in the **round after it receives the bit**. To allow S to stop sending the bit, R continues to send the ack repeatedly from then on.

This informal description gives what we call a **protocol for S and R** : it is a specification of what they do at each step.

Bit-transmission problem

- ▶ S sends the bit to R until S receives the ack message; before it receives the ack message, S **does not know** whether R received the bit.
- ▶ R **knows** perfectly well that S stops sending messages after it receives an ack message, but R **never knows** for certain that S actually received its acknowledgment.
- ▶ Even if R does not receive messages from S for a while, R **does not know** whether this is because S received an ack message from R ; it could be because the messages that S sent were lost in the communication channel.

301

Bit-transmission problem

- ▶ We could have S send an **ack-ack** message (an acknowledgment to the acknowledgment) so that R could stop sending the acknowledgment once it receives an ack-ack message from S .
- ▶ But this only pushes the problem up one level: S will not be able to safely stop sending ack-ack messages, since S has no way of **knowing** that R has received an ack-ack message.
- ▶ This type of **uncertainty is inherent** in systems where communication is not guaranteed.

302

Bit-transmission problem

We formalize the bit-transmission problem as a **system**.

To describe the **set of runs** that make up this system,

- ▶ we choose to have the local states of S and R include very little information; essentially, just enough to allow us to carry out our analysis.
- ▶ it is useful to have the environment's state record the events taking place in the system.

303

Bit-transmission problem

Let L_S be the set of possible local states of S . We take

$$L_S = \{0, 1, (0, \text{ack}), (1, \text{ack})\}.$$

- ▶ S 's local state is $k \in \{0, 1\}$ if its initial bit is k and it has not received an ack message from R .
- ▶ S 's local state is (k, ack) if its initial bit is $k \in \{0, 1\}$ and it has received an ack message from R .

Let L_R be the set of possible local states of R . We take

$$L_R = \{\lambda, 0, 1\}.$$

- ▶ λ denotes the local state where R has received no messages from S .
- ▶ $k \in \{0, 1\}$ denotes the local state where R received the message k from S .

304

Bit-transmission problem

The environment's local state is used to record the history of events taking place in the system. At each round, we have the following possibilities:

- ▶ S sends the bit to R and R does nothing.
Notation: $(sendbit, \Lambda)$.
- ▶ S does nothing and R sends an ack to S .
Notation: $(\Lambda, sendack)$.
- ▶ both S and R send messages. Notation: $(sendbit, sendack)$.

We let the environment's state be a finite sequence of elements from the set

$$\{(sendbit, \Lambda), (\Lambda, sendack), (sendbit, sendack)\}.$$

Here the m^{th} member of the sequence describes the actions of the sender and receiver in round m .

305

Bit-transmission problem

There are many possible runs in this system, but these runs must all satisfy certain constraints.

Initially, the system must start in a global state where nothing has been recorded in the environment's state, neither S nor R has received any messages, and S has an initial bit of either 0 or 1. Thus, the initial global state of every run in the system has the form

$$(\(), k, \lambda),$$

where $()$ is the empty sequence and $k \in \{0, 1\}$.

306

Bit-transmission problem

Consecutive global states

$$r(m) = (s_e, s_S, s_R) \text{ and } r(m+1) = (s'_e, s'_S, s'_R) \text{ in a run } r$$

are related by the following conditions:

- ▶ If $s_R = \lambda$, then $s'_S = s_S$, $s'_e = s_e(sendbit, \Lambda)$ (where $s_e(sendbit, \Lambda)$ is the result of appending $(sendbit, \Lambda)$ to the sequence s_e), and either $s'_R = \lambda$ or $s'_R = s_S$.

Before R receives a message, it sends no messages; as a result, S receives no message, so it continues to send the bit and its state does not change. R may or may not receive the message sent by S in round $m+1$.

307

Bit-transmission problem

Consecutive global states

$$r(m) = (s_e, s_S, s_R) \text{ and } r(m+1) = (s'_e, s'_S, s'_R) \text{ in a run } r$$

are related by the following conditions:

- ▶ If $s_S = s_R = k$, then $s'_R = k$, $s'_e = s_e(sendbit, sendack)$, and either $s'_S = k$ or $s'_S = (k, ack)$.

After R has received S 's bit, it starts sending acknowledgments, and its state undergoes no further changes. S continues to send the bit, and it may or may not receive the acknowledgment sent by R in round $m+1$.

308

Bit-transmission problem

Consecutive global states

$r(m) = (s_e, s_S, s_R)$ and $r(m+1) = (s'_e, s'_S, s'_R)$ in a run r

are related by the following conditions:

- ▶ If $s_S = (k, ack)$, then $s'_e = s_e(\Lambda, sendack)$, $s'_S = s_S$, and $s'_R = s_R$.

Once S has received R 's acknowledgment, S stops sending the bit and R continues to send acknowledgments. The local states of S and R do not change any more.

Definition 6.4

We take the system \mathcal{R}^{bt} describing the *bit-transmission problem* to consist of all the runs meeting the constraints just described.

309

Incorporating knowledge and time

310

Incorporating knowledge

- ▶ We already saw in the bit-transmission problem that we were making statements such as “ R **does not know** for certain that S received its acknowledgment.”
- ▶ An agent's actions depend on its knowledge.
- ▶ We shall see that **knowledge** can be incorporated in our framework in a straightforward way.
- ▶ The basic idea is that a statement such as **R does not know φ** means that
 - (*) as far as R is concerned, the system could be at a point where φ does not hold.
- ▶ We capture (*) using **frames for epistemic logic**.
- ▶ We think of R 's **knowledge** as being determined by its **local state**, so that R cannot distinguish between two points of the system in which it has the same local state, and it can distinguish points in which its local state differs.

311

Incorporating knowledge

Let \mathcal{R} be a system over \mathcal{G} and i be an agent. Recall that $\mathcal{P}_{\mathcal{R}}$ denotes the set of points of \mathcal{R} .

Definition 6.5

We define the binary relations \sim_i on \mathcal{G} and \mathcal{K}_i on $\mathcal{P}_{\mathcal{R}}$ as follows:

- ▶ for all global states $s = (s_e, s_1, \dots, s_n)$, $s' = (s'_e, t'_1, \dots, t'_n)$,

$$s \sim_i s' \quad \text{iff} \quad s_i = s'_i.$$

- ▶ for all points (r, m) , (r', m') ,

$$(r, m) \mathcal{K}_i (r', m') \quad \text{iff} \quad r(m) \sim_i r'(m') \quad \text{iff} \quad r_i(m) = r'_i(m').$$

If $s \sim_i s'$ or $(r, m) \mathcal{K}_i (r', m')$, we say that they are **indistinguishable to agent i** .

312

- ▶ For every agent i , \sim_i and \mathcal{K}_i are equivalence relations.
- ▶ $\mathcal{F}_{\mathcal{R}} = (\mathcal{P}_{\mathcal{R}}, \mathcal{K}_1, \dots, \mathcal{K}_n)$ is a frame for the epistemic logic **S5**.

Thus, to every system \mathcal{R} we associate a frame $\mathcal{F}_{\mathcal{R}}$ for the epistemic logic **S5**.

Note that there is no relation \mathcal{K}_e for the environment. This is because we are not usually interested in what the environment knows.

Let $PROP$ be a set of atomic propositions.

Definition 6.6

An **interpreted system** is a pair $\mathcal{I} = (\mathcal{R}, \pi)$, where \mathcal{R} is a system over a set \mathcal{G} of global states and $\pi : \mathcal{G} \rightarrow (PROP \rightarrow \{0, 1\})$ is an **interpretation**.

We also say that \mathcal{I} is **based** on \mathcal{R} or that \mathcal{R} is the system **underlying** \mathcal{I} .

Thus, π assigns **truth values** to atomic propositions at global states: for every state $s \in \mathcal{G}$ and $p \in PROP$, $\pi(s)(p) \in \{0, 1\}$.

Remark 6.7

π induces also an interpretation over the points of \mathcal{R} . For every point $(r, m) \in \mathcal{R}$, take $\pi((r, m))$ to be $\pi(r(m))$. That is, for every $p \in PROP$,

$$\pi((r, m))(p) = \pi(r(m))(p).$$

We refer to the points and states of the system \mathcal{R} as points and states, respectively, of the interpreted system \mathcal{I} . That is, we shall also use the notation $\mathcal{P}_{\mathcal{I}}$ for $\mathcal{P}_{\mathcal{R}}$.

Thus,

- ▶ (r, m) is a point in the interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ iff (r, m) is a point in \mathcal{R} iff $r \in \mathcal{R}$.
- ▶ \mathcal{I} is an interpreted system over \mathcal{G} if \mathcal{R} is a system over \mathcal{G} .

Let $\mathcal{I} = (\mathcal{R}, \pi)$ be an interpreted system.

Define $V_{\mathcal{I}} : PROP \rightarrow 2^{\mathcal{P}_{\mathcal{I}}}$ as follows: for all $p \in PROP$,

$$\begin{aligned} V_{\mathcal{I}}(p) &= \{(r, m) \in \mathcal{P}_{\mathcal{I}} \mid \pi((r, m))(p) = 1\} \\ &= \{(r, m) \in \mathcal{P}_{\mathcal{I}} \mid \pi(r(m))(p) = 1\}. \end{aligned}$$

$\mathcal{M}_{\mathcal{I}} = (\mathcal{P}_{\mathcal{I}}, V_{\mathcal{I}}, \mathcal{K}_1, \dots, \mathcal{K}_n)$ is a model for the epistemic logic **S5**.

Thus, to every interpreted system \mathcal{I} we associate a model $\mathcal{M}_{\mathcal{I}}$ for the epistemic logic **S5**.

Definition 6.8

For every formula φ of ML_{AG} , we say that

$$\varphi \text{ is true in } \mathcal{I} \text{ at point } (r, m) \quad \text{iff} \quad \mathcal{M}_{\mathcal{I}}, (r, m) \Vdash \varphi.$$

Notation: $(\mathcal{I}, r, m) \models \varphi$.

Remark 6.9

For every $p \in PROP$, $(\mathcal{I}, r, m) \models p$ iff $\mathcal{M}_{\mathcal{I}}, (r, m) \Vdash p$ iff $(r, m) \in V_{\mathcal{I}}(p)$ iff $\pi((r, m))(p) = 1$ iff $\pi(r(m))(p) = 1$.

As π is a function on global states, the truth of an atomic proposition p at a point (r, m) depends only on the global state $r(m)$. This seems like a natural assumption; the global state is meant to capture everything that is relevant about the current situation.

Remark 6.10

For every $i \in AG$ and every formula φ , $(\mathcal{I}, r, m) \models K_i \varphi$

iff $\mathcal{M}_{\mathcal{I}}, (r, m) \Vdash K_i \varphi$

iff $\mathcal{M}_{\mathcal{I}}, (r', m') \Vdash \varphi$ for all (r', m') such that $(r, m) \mathcal{K}_i (r', m')$

iff $\mathcal{M}_{\mathcal{I}}, (r', m') \Vdash \varphi$ for all (r', m') such that $r(m) \sim_i r'(m')$

iff $\mathcal{M}_{\mathcal{I}}, (r', m') \Vdash \varphi$ for all (r', m') such that $r_i(m) = r'_i(m')$.

Proposition 6.11

Let (r, m) and (r', m') be points in \mathcal{I} such that $r(m) = r'(m')$. Then for every formula φ ,

$$(\mathcal{I}, r, m) \models \varphi \quad \text{iff} \quad (\mathcal{I}, r', m') \models \varphi.$$

Proof: By induction on φ .

- ▶ $\varphi = p \in PROP$. Then $(\mathcal{I}, r, m) \models p$ iff $\pi(r(m))(p) = 1$ iff $\pi(r'(m'))(p) = 1$ iff $(\mathcal{I}, r', m') \models \varphi$.
- ▶ The cases $\varphi = \neg\psi$ and $\varphi = \psi \rightarrow \chi$ are obvious.
- ▶ $\varphi = K_i \psi$. Then
 - $(\mathcal{I}, r, m) \models K_i \psi$ iff $\mathcal{M}_{\mathcal{I}}, (r^*, m^*) \Vdash \psi$ for all (r^*, m^*) such that $r_i(m) = r'_i(m^*)$
 - iff $\mathcal{M}_{\mathcal{I}}, (r^*, m^*) \Vdash \psi$ for all (r^*, m^*) such that $r'_i(m') = r'_i(m^*)$
 - iff $(\mathcal{I}, r', m') \models K_i \psi$. □

Definition 6.12

We say that φ is **true** in an interpreted system \mathcal{I} if $(\mathcal{I}, r, m) \models \varphi$ for all points (r, m) in \mathcal{I} .

Notation: $\mathcal{I} \models \varphi$.

Definition 6.13

Let \mathcal{M} be a class of interpreted systems. We say that φ is **true** in \mathcal{M} if $\mathcal{I} \models \varphi$ for every interpreted system $\mathcal{I} \in \mathcal{M}$.

Notation: $\mathcal{M} \models \varphi$.

Definition 6.14

We say that φ is **valid** in a system \mathcal{R} if $\mathcal{I} \models \varphi$ for every interpreted system \mathcal{I} based on \mathcal{R} .

Notation: $\mathcal{I} \models \varphi$.

Definition 6.15

Let \mathcal{F} be a class of systems. We say that φ is **valid** in \mathcal{F} if $\mathcal{R} \models \varphi$ for every system $\mathcal{R} \in \mathcal{F}$.

Notation: $\mathcal{F} \models \varphi$.

321

Consider the bit-transmission problem again.

We take *PROP* to consist of six atomic propositions:

- ▶ **bit = 0** representing the assertion that **the value of S's initial bit is 0**;
- ▶ **bit = 1** representing the assertion that **the value of S's initial bit is 1**;
- ▶ **recbit** representing the assertion that **R has received S's message**;
- ▶ **recack** representing the assertion that **S has received R's acknowledgment**;
- ▶ **sentbit** representing the assertion that **S has just sent a message**;
- ▶ **sentack** representing the assertion that **R has just sent a message**.

322

Definition 6.16

Define the interpreted system $\mathcal{I}^{bt} = (\mathcal{R}^{bt}, \pi^{bt})$, where \mathcal{R}^{bt} is the system defined by Definition 6.4 and π^{bt} is an interpretation such that for all points (r, m) ,

- ▶ $(\mathcal{I}^{bt}, r, m) \models \text{bit} = k$ iff $r_S(m)$ is either k or (k, ack) for $k = 0, 1$;
- ▶ $(\mathcal{I}^{bt}, r, m) \models \text{recbit}$ iff $r_R(m) \neq \lambda$;
- ▶ $(\mathcal{I}^{bt}, r, m) \models \text{recack}$ iff $r_S(m) = (k, \text{ack})$ for some $k = 0, 1$;
- ▶ $(\mathcal{I}^{bt}, r, m) \models \text{sentbit}$ iff the last tuple in $r_e(m)$ is $(\text{sendbit}, \text{sendack})$ or $(\text{sendbit}, \Lambda)$;
- ▶ $(\mathcal{I}^{bt}, r, m) \models \text{sentack}$ iff the last tuple in $r_e(m)$ is $(\text{sendbit}, \text{sendack})$ or $(\Lambda, \text{sendack})$.

The truth value of all the atomic propositions is **completely determined by the global state**, since the environment's state records the events taking place in the system.

323

After R receives S 's bit, then R **knows** the value of the bit.

Proposition 6.17

Let (r, m) be a point such that $r_R(m) = k$ for $k = 0, 1$. Then $(\mathcal{I}^{bt}, r, m) \models K_R(\text{bit} = k)$.

Proof: Let (r', m') be a point s.t. $r_R(m) = r'_R(m') = k$. Then S must have initial bit k at (r', m') , so $(\mathcal{I}^{bt}, r', m') \models \text{bit} = k$. \square

When S receives R 's ack message, then S knows that R knows the initial bit.

Proposition 6.18

Let (r, m) be a point such that $r_S(m) = (k, \text{ack})$ for $k = 0, 1$. Then $(\mathcal{I}^{bt}, r, m) \models K_S K_R(\text{bit} = k)$.

Proof: Let (r', m') be a point s.t. $r_S(m) = r'_S(m') = (k, \text{ack})$. We have to prove that $(\mathcal{I}^{bt}, r', m') \models K_R(\text{bit} = k)$. This follows from Proposition 6.17, using the fact that $r'_R(m') = k$. \square

324

Incorporating time

- ▶ The **temporal language** ML_{temp} is obtained by adding to ML_{Ag} the following temporal operators: \Box (**always**), its dual \Diamond (**eventually**), \bigcirc (**next time**), and U (**until**).
- ▶ Formulas of ML_{temp} are defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid K_i\varphi \mid \Box\varphi \mid \Diamond\varphi \mid \bigcirc\varphi \mid \varphi U\psi,$$
 where $p \in PROP$ and $i \in Ag$.
- ▶ We sometimes refer to formulas of ML_{temp} as **temporal formulas** and to formulas of ML_{Ag} as **knowledge formulas**.

Intuition:

- ▶ $\Box\varphi$ is true if φ is true now and at all later points;
- ▶ $\Diamond\varphi$ is true if φ is true at some point in the future;
- ▶ $\bigcirc\varphi$ is true if φ is true at the next step;
- ▶ $\varphi U\psi$ is true if φ is true until ψ is true.

325

Incorporating time

Let $\mathcal{I} = (\mathcal{R}, \pi)$ be an interpreted system. Then for all points (r, m) in \mathcal{I} and every formulas φ, ψ ,

$$\begin{aligned} (\mathcal{I}, r, m) \models \Box\varphi & \text{ iff } (\mathcal{I}, r, m') \models \varphi \text{ for all } m' \geq m, \\ (\mathcal{I}, r, m) \models \Diamond\varphi & \text{ iff } (\mathcal{I}, r, m') \models \varphi \text{ for some } m' \geq m, \\ (\mathcal{I}, r, m) \models \bigcirc\varphi & \text{ iff } (\mathcal{I}, r, m+1) \models \varphi, \\ (\mathcal{I}, r, m) \models \varphi U\psi & \text{ iff } (\mathcal{I}, r, m') \models \psi \text{ for some } m' \geq m \text{ and} \\ & (\mathcal{I}, r, m'') \models \varphi \text{ for all } m'' \text{ with } m \leq m'' < m'. \end{aligned}$$

- ▶ The interpretation of $\bigcirc\varphi$ as “ φ is true at the next step” makes sense because our notion of time is discrete.
- ▶ All the other temporal operators make perfect sense even for continuous notions of time.

326

Incorporating time

Proposition 6.19

For every interpreted system \mathcal{I} ,

$$\mathcal{I} \models \Diamond\varphi \leftrightarrow \top U\varphi \text{ and } \mathcal{I} \models \Box\varphi \leftrightarrow \neg\Diamond\neg\varphi.$$

Thus, we can take \bigcirc and U as our basic temporal operators, and define \Diamond and \Box in terms of U .

- ▶ The truth of a temporal formula depends only on the run.
- ▶ The truth of φ at a point (r, m) in $\mathcal{I} = (\mathcal{R}, \pi)$ **does not depend on \mathcal{R} at all**, but **only on π** , so we can write $(\pi, r, m) \models \varphi$.
- ▶ In general, temporal operators are used for reasoning about events that happen along a single run.

Definition 6.20

We say that r **satisfies φ** if $(\pi, r, 0) \models \varphi$ holds.

327

Incorporating time

Once we have temporal operators, there are a number of important notions that we can express.

- ▶ The formula $\Box\Diamond\varphi$ is true iff φ **occurs infinitely often**; that is, $(\mathcal{I}, r, m) \models \Box\Diamond\varphi$ exactly if the set $\{m' \mid (\mathcal{I}, r, m') \models \varphi\}$ is infinite.
- ▶ The formula $\Diamond\Box\varphi$ is true iff φ **is true almost everywhere**; that is, $(\mathcal{I}, r, m) \models \Diamond\Box\varphi$ iff for some m' and all $m'' \geq m'$, we have $(\mathcal{I}, r, m'') \models \varphi$.
- ▶ The temporal operators that we have defined can talk about events that happen only in the present or future.
- ▶ We can add temporal operators for reasoning about the past, for example, an analogue to \Diamond that says **at some time in the past**.

328



Bit-transmission problem - again

Consider the bit-transmission problem again.

The formula $\Box(\text{recbit} \rightarrow \Diamond \text{recack})$ says that if at some point along a run the receiver receives the bit sent by the sender, then at some point in the future the sender will receive the acknowledgment sent by the receiver.

The formula $\Diamond(K_R(\text{bit} = 0) \vee K_R(\text{bit} = 1))$ says that the receiver eventually knows the sender's initial bit.