Logic for Multiagent Systems

Master 1st Year, 1st Semester 201/2022

Laurențiu Leuștean

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

Propositional logic

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

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

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

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

- *If* $0 \leq i \leq j \leq k-1$, *then the expression* $\theta_i \ldots \theta_j$ *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* $Var(\theta)$ *the set of variables appearing in* θ.

Language

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 $(\varphi \to \psi)$ 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 $\varphi, \psi, \chi, \ldots$.

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.
- $\varphi = (\psi \to \chi)$, 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 $\Gamma = Form$.

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 $\Gamma = Form$.

Language

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

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

Conventions and notations

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

Truth values

We use the following notations for the truth values:

$$1 \text{ for true and } 0 \text{ for false.}$$

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

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

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

p	$\neg p$
0	1
1	0

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

p	q	$p \to q$
0	0	1
0	1	1
1	0	0
1	1	1

Semantics

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

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

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

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

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

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

Definition 1.8

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

Theorem 1.9

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

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

satisfying the following properties:

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

Proposition 1.10

For any formula φ *and all evaluations* $e_1, e_2 : V \to \{0, 1\}$,

if $e_1(v) = e_2(v)$ *for all* $v \in Var(\varphi)$, *then* $e_1^+(\varphi) = e_2^+(\varphi)$.

Semantics

Let φ be a formula.

Definition 1.11

- *An evaluation $e : V \to \{0, 1\}$ 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 $Mod(\varphi)$.

Semantics

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

- $e \vDash \neg\varphi$ iff $e \nvDash \varphi$.
- $e \vDash \varphi \to \psi$ iff ($e \vDash \varphi$ implies $e \vDash \psi$) iff ($e \nvDash \varphi$ or $e \vDash \psi$).
- $e \vDash \varphi \vee \psi$ iff ($e \vDash \varphi$ or $e \vDash \psi$).
- $e \vDash \varphi \wedge \psi$ iff ($e \vDash \varphi$ and $e \vDash \psi$).
- $e \vDash \varphi \leftrightarrow \psi$ iff ($e \vDash \varphi$ iff $e \vDash \psi$).

Definition 1.15

Let φ, ψ be formulas. We say that

- φ is a semantic consequence of ψ if $Mod(\psi) \subseteq Mod(\varphi)$.
  Notation: $\psi \vDash \varphi$.
- φ and ψ are (logically) equivalent if $Mod(\psi) = Mod(\varphi)$.
  Notation: $\varphi \sim \psi$.

Remark 1.16

Let φ, ψ be formulas.

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

Semantics

For all formulas φ, ψ, χ,

$$
\begin{aligned}
&\models \quad \varphi \vee \neg\varphi \\
&\models \quad \neg(\varphi \wedge \neg\varphi) \\
&\models \quad \varphi \wedge \psi \to \varphi \\
&\models \quad \varphi \to \varphi \vee \psi \\
&\models \quad \varphi \to (\psi \to \varphi) \\
&\models \quad (\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)) \\
&\models \quad (\varphi \to \psi) \to ((\psi \to \chi) \to (\varphi \to \chi)) \\
&\models \quad (\varphi \to \psi) \vee (\neg\varphi \to \psi) \\
&\models \quad (\varphi \to \psi) \vee (\varphi \to \neg\psi) \\
&\models \quad \neg\varphi \to (\neg\psi \leftrightarrow (\psi \to \varphi)) \\
&\models \quad (\varphi \to \psi) \to (((\varphi \to \chi) \to \psi) \to \psi) \\
&\models \quad \neg\psi \to (\psi \to \varphi)
\end{aligned}
$$

Semantics

$$
\begin{array}{rcl}
 & \models & \psi \to (\neg\psi \to \varphi) \\
 & \models & (\varphi \to \neg\varphi) \to \neg\varphi \\
 & \models & (\neg\varphi \to \varphi) \to \varphi \\
\psi & \models & \varphi \to \psi \\
\neg\varphi & \models & \varphi \to \psi \\
\neg\psi \wedge (\varphi \to \psi) & \models & \neg\varphi \\
(\varphi \to \psi) \wedge (\psi \to \chi) & \models & \varphi \to \chi \\
\varphi \wedge (\varphi \to \psi) & \models & \psi \\
\{\psi, \neg\psi\} & \models & \varphi \\
\{\psi, \neg\varphi\} & \models & \neg(\psi \to \varphi)
\end{array}
$$

Semantics

$$
\begin{array}{rcl}
\varphi & \sim & \neg\neg\varphi \\
\varphi \to \psi & \sim & \neg\psi \to \neg\varphi \\
\varphi \vee \psi & \sim & \neg(\neg\varphi \wedge \neg\psi) \\
\varphi \wedge \psi & \sim & \neg(\neg\varphi \vee \neg\psi) \\
\varphi \to (\psi \to \chi) & \sim & \varphi \wedge \psi \to \chi \\
\varphi \sim \varphi \wedge \varphi & \sim & \varphi \vee \varphi \\
\varphi \wedge \psi & \sim & \psi \wedge \varphi \\
\varphi \vee \psi & \sim & \psi \vee \varphi \\
\varphi \wedge (\psi \wedge \chi) & \sim & (\varphi \wedge \psi) \wedge \chi \\
\varphi \vee (\psi \vee \chi) & \sim & (\varphi \vee \psi) \vee \chi \\
\varphi \vee (\varphi \wedge \psi) & \sim & \varphi \\
\varphi \wedge (\varphi \vee \psi) & \sim & \varphi
\end{array}
$$

Semantics

$$
\begin{array}{rcl}
\varphi \wedge (\psi \vee \chi) & \sim & (\varphi \wedge \psi) \vee (\varphi \wedge \chi) \\
\varphi \vee (\psi \wedge \chi) & \sim & (\varphi \vee \psi) \wedge (\varphi \vee \chi) \\
\varphi \to \psi \wedge \chi & \sim & (\varphi \to \psi) \wedge (\varphi \to \chi) \\
\varphi \to \psi \vee \chi & \sim & (\varphi \to \psi) \vee (\varphi \to \chi) \\
\varphi \wedge \psi \to \chi & \sim & (\varphi \to \chi) \vee (\psi \to \chi) \\
\varphi \vee \psi \to \chi & \sim & (\varphi \to \chi) \wedge (\psi \to \chi) \\
\varphi \to (\psi \to \chi) & \sim & \psi \to (\varphi \to \chi) \\
 & \sim & (\varphi \to \psi) \to (\varphi \to \chi) \\
\neg\varphi \sim \varphi \to \neg\varphi & \sim & (\varphi \to \psi) \wedge (\varphi \to \neg\psi) \\
\varphi \to \psi \sim \neg\varphi \vee \psi & \sim & \neg(\varphi \wedge \neg\psi) \\
\varphi \vee \psi \sim \varphi \vee (\neg\varphi \wedge \psi) & \sim & (\varphi \to \psi) \to \psi \\
\varphi \leftrightarrow (\psi \leftrightarrow \chi) & \sim & (\varphi \leftrightarrow \psi) \leftrightarrow \chi
\end{array}
$$

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

Remark 1.17

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

Notation 1.18

Denote $v_0 \to v_0$ by $\top$ and call it the truth.
Denote $\neg(v_0 \to v_0)$ by $\bot$ and call it the false.

Remark 1.19

- *φ is a tautology iff $\varphi \sim \top$.*
- *φ is unsatisfiable iff $\varphi \sim \bot$.*

Let Γ be a set of formulas.

Definition 1.20

An evaluation $e : V \to \{0, 1\}$ is a model of Γ if it is a model of every formula from Γ.
Notation: $e \vDash \Gamma$.

Notation 1.21

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

Definition 1.22

A formula φ is a semantic consequence of Γ if $Mod(\Gamma) \subseteq Mod(\varphi)$.
Notation: $\Gamma \vDash \varphi$.

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.*
- $\Gamma \models \bot$.

Theorem 1.25 (Compactness Theorem)

Γ *is satisfiable iff* Γ *is finitely satisfiable.*

Syntax

We use a deductive system of Hilbert type for LP.

Logical axioms

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

(A1) $\varphi \to (\psi \to \varphi)$

(A2) $(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$

(A3) $(\neg\psi \to \neg\varphi) \to (\varphi \to \psi)$,

where φ, ψ and χ are formulas.

The deduction rule

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

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

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 $\varphi \to \psi$ 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.

Notations

$\Gamma \vdash \varphi \quad :\Leftrightarrow \quad \varphi$ is a Γ-theorem

$\vdash \varphi \quad :\Leftrightarrow \quad \emptyset \vdash \varphi$.

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 $\Gamma \vdash \varphi \to \psi$, then $\Gamma \vdash \psi$.*

Definition 1.29

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

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

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 $\theta_1, \ldots, \theta_n$ such that $\theta_n = \varphi$.

Proposition 1.31

For any formula φ,

$$\Gamma \vdash \varphi \quad \text{iff} \quad \text{there exists a } \Gamma\text{-proof of } \varphi.$$

Theorem 1.32 (Deduction Theorem)

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

$$\Gamma \cup \{\varphi\} \vdash \psi \quad \textit{iff} \quad \Gamma \vdash \varphi \to \psi.$$

Proposition 1.33

For any formulas φ, ψ, χ*,*

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

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

Proposition 1.34

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

$$\Gamma \vdash \varphi \to \psi \text{ and } \Gamma \vdash \psi \to \chi \quad \Rightarrow \quad \Gamma \vdash \varphi \to \chi$$

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

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

Consistent sets

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.*
- *$\Gamma \vdash \bot$.*

Theorem 1.38 (Completeness Theorem (version 1))

Let Γ *be a set of formulas. Then*

$$\Gamma \text{ is consistent} \iff \Gamma \text{ is satisfiable.}$$

Theorem 1.39 (Completeness Theorem (version 2))

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

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

First-order logic

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 $\rightarrow$;*
- *parantheses (,);*
- *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 $\mathrm{ari} : \mathcal{F} \cup \mathcal{R} \rightarrow \mathbb{N}^*$.*

- $\mathcal{L}$ is uniquely determined by the quadruple $\tau := (\mathcal{R}, \mathcal{F}, \mathcal{C}, \mathrm{ari})$.
- τ is called the signature of $\mathcal{L}$ or the similaritaty type of $\mathcal{L}$.

First-order languages

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

- The set $Sym_{\mathcal{L}}$ of symbols of $\mathcal{L}$ is

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

- 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 $x, y, z, v, \ldots$, relation symbols by $P, Q, R \ldots$, function symbols by $f, g, h, \ldots$ and constant symbols by $c, d, e, \ldots$.

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

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

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

Definition 2.2

The set $Expr_{\mathcal{L}}$ of expressions of $\mathcal{L}$ is the set of all finite sequences of symbols of $\mathcal{L}$.

Definition 2.3

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

- *If $0 \leq i \leq j \leq k-1$, then the expression $\theta_i \ldots \theta_j$ 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 $Var(\theta)$ the set of variables appearing in θ.*

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 $t_1, \ldots, t_m$ are terms, then $ft_1 \ldots t_m$ 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 $t, s, t_1, t_2, s_1, s_2, \ldots$.
- $Var(t)$ 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 $t_1, \ldots, t_m \in \Gamma$, then $ft_1 \ldots t_m \in \Gamma$.*

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;
- $(Rt_1 \ldots t_m)$, where $R \in \mathcal{R}_m$ and $t_1, \ldots, t_m$ 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 $(\varphi \to \psi)$ 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 $\varphi, \psi, \chi, \ldots$.
- $Var(\varphi)$ 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.
- $\varphi = (Rt_1 \ldots t_m)$, where $R \in \mathcal{R}_m$ and $t_1, \ldots, t_m$ are terms.
- $\varphi = (\neg\psi)$, where ψ is a formula.
- $\varphi = (\psi \to \chi)$, 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:*

$$\text{if } \varphi, \psi \in \Gamma, \text{ then } (\neg\varphi), (\varphi \rightarrow \psi), (\forall x \varphi) \in \Gamma.$$

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

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 $\Gamma = Form_{\mathcal{L}}$.

Derived connectives

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

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

First-order languages

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

To reduce the use of parentheses, we assume that

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

- We write sometimes $f(t_1, \ldots, t_m)$ instead of $ft_1 \ldots t_m$ and $R(t_1, \ldots, t_m)$ instead of $Rt_1 \ldots t_m$.
- 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 $t_1 f t_2$ instead of $ft_1 t_2$.
- If R is a binary relation symbol, we write $t_1 R t_2$ instead of $Rt_1 t_2$.

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 $\varphi = \varphi_0\varphi_1 \ldots \varphi_{n-1}$ 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 $0 \le i \le k \le j \le n-1$ 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 $\varphi = \forall x(x = y) \to x = z$. Free variables: x, y, z. Bound variables: x.

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:

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

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* $f^{\mathcal{A}} : A^m \to A$.
- $\mathcal{R}^{\mathcal{A}} = \{R^{\mathcal{A}} \mid R \in \mathcal{R}\}$ *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* $\mathcal{A}$.

Examples - The language of equality $\mathcal{L}_=$

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

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

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

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

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

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

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

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

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

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

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

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 $\dot{\leq}$ 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 $\dot{<}$ 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 $\dot{\in}$ instead of R and we denote the language by $\mathcal{L}_{\in}$.

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 $e : V \to A$.

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

In the following, $e : V \to A$ 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 $t = ft_1 \ldots t_m$, then $t^{\mathcal{A}}(e) := f^{\mathcal{A}}(t_1^{\mathcal{A}}(e), \ldots, t_m^{\mathcal{A}}(e))$.*

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.

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

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

Negation and implication

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

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

p	q	$p \to q$
0	0	1
0	1	1
1	0	0
1	1	1

Hence,

- $(\neg\varphi)^{\mathcal{A}}(e) = 1$ iff $\varphi^{\mathcal{A}}(e) = 0$.
- $(\varphi \to \psi)^{\mathcal{A}}(e) = 1$ iff $\left(\varphi^{\mathcal{A}}(e) = 0 \text{ or } \psi^{\mathcal{A}}(e) = 1 \right)$.

Semantics

Notation

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

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

Universal quantifier

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

Semantics

Let $\mathcal{A}$ be an $\mathcal{L}$-structure and $e : V \to A$ 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: $\mathcal{A} \vDash \varphi[e]$.*
- *e does not satisfy φ in $\mathcal{A}$ if $\varphi^{\mathcal{A}}(e) = 0$. Notation: $\mathcal{A} \nvDash \varphi[e]$.*

Proposition 2.15

For all formulas φ, ψ and any variable x,

(i) $\mathcal{A} \vDash \neg\varphi[e]$ *iff* $\mathcal{A} \nvDash \varphi[e]$.

(ii) $\mathcal{A} \vDash (\varphi \to \psi)[e]$ *iff* ($\mathcal{A} \vDash \varphi[e]$ *implies* $\mathcal{A} \vDash \psi[e]$)
iff ($\mathcal{A} \nvDash \varphi[e]$ *or* $\mathcal{A} \vDash \psi[e]$).

(iii) $\mathcal{A} \vDash (\forall x\varphi)[e]$ *iff for all* $a \in A$, $\mathcal{A} \vDash \varphi[e_{x \leftarrow a}]$.

Proposition 2.16

For all formulas φ, ψ and any variable x,

(i) $\mathcal{A} \vDash (\varphi \wedge \psi)[e]$ *iff* ($\mathcal{A} \vDash \varphi[e]$ *and* $\mathcal{A} \vDash \psi[e]$).

(ii) $\mathcal{A} \vDash (\varphi \vee \psi)[e]$ *iff* ($\mathcal{A} \vDash \varphi[e]$ *or* $\mathcal{A} \vDash \psi[e]$).

(iii) $\mathcal{A} \vDash (\varphi \leftrightarrow \psi)[e]$ *iff* ($\mathcal{A} \vDash \varphi[e]$ *iff* $\mathcal{A} \vDash \psi[e]$).

(iv) $\mathcal{A} \vDash (\exists x \varphi)[e]$ *iff there exists* $a \in A$ *s.t.* $\mathcal{A} \vDash \varphi[e_{x \leftarrow a}]$.

Semantics

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: $\models \varphi$

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,

$$\mathcal{A} \vDash \varphi[e] \quad \textit{implies} \quad \mathcal{A} \vDash \psi[e].$$

Notation: $\varphi \vDash \psi$

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,

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

Notation: $\varphi \mathrel{\text{⫤}\!\vDash} \psi$

Remark

- $\varphi \vDash \psi$ iff $\vDash \varphi \rightarrow \psi$.
- $\varphi \mathrel{\text{⫤}\!\vDash} \psi$ iff ($\psi \vDash \varphi$ and $\varphi \vDash \psi$) iff $\vDash \psi \leftrightarrow \varphi$.

Semantics

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

$$\neg\exists x\varphi \quad \mathrel{=}\!\!\!\mid\!\!\models \quad \forall x\neg\varphi \tag{1}$$

$$\neg\forall x\varphi \quad \mathrel{=}\!\!\!\mid\!\!\models \quad \exists x\neg\varphi \tag{2}$$

$$\forall x(\varphi \wedge \psi) \quad \mathrel{=}\!\!\!\mid\!\!\models \quad \forall x\varphi \wedge \forall x\psi \tag{3}$$

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

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

$$\exists x(\varphi \vee \psi) \quad \mathrel{=}\!\!\!\mid\!\!\models \quad \exists x\varphi \vee \exists x\psi \tag{6}$$

$$\forall x(\varphi \to \psi) \quad \models \quad \forall x\varphi \to \forall x\psi \tag{7}$$

$$\forall x(\varphi \to \psi) \quad \models \quad \exists x\varphi \to \exists x\psi \tag{8}$$

$$\forall x\varphi \quad \models \quad \exists x\varphi \tag{9}$$

Semantics

$$
\begin{aligned}
\varphi &\models \exists x \varphi && (10)\\
\forall x \varphi &\models \varphi && (11)\\
\forall x \forall y \varphi &\mathrel{=\!\!\mid\!\models} \forall y \forall x \varphi && (12)\\
\exists x \exists y \varphi &\mathrel{=\!\!\mid\!\models} \exists y \exists x \varphi && (13)\\
\exists y \forall x \varphi &\models \forall x \exists y \varphi. && (14)
\end{aligned}
$$

Proposition 2.22

For all terms s, t, u,

(i) $\models t = t$;

(ii) $\models s = t \to t = s$;

(iii) $\models s = t \wedge t = u \to s = u$.

Proposition 2.23

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

$$\models (t_1 = u_1) \wedge \ldots \wedge (t_m = u_m) \to ft_1 \ldots t_m = fu_1 \ldots u_m$$

$$\models (t_1 = u_1) \wedge \ldots \wedge (t_m = u_m) \to (Rt_1 \ldots t_m \leftrightarrow Ru_1 \ldots u_m)$$

Proposition 2.24

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

(i) *for any term t,*

$$\text{if } e_1(v) = e_2(v) \text{ for all variables } v \in Var(t), \text{ then } t^{\mathcal{A}}(e_1) = t^{\mathcal{A}}(e_2).$$

(ii) *for any formula φ,*

$$\text{if } e_1(v) = e_2(v) \text{ for all variables } v \in FV(\varphi), \text{ then } \mathcal{A} \models \varphi[e_1] \text{ iff } \mathcal{A} \models \varphi[e_2].$$

Proposition 2.25

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

$$\varphi \;\Dashv\vDash\; \exists x\varphi \tag{15}$$

$$\varphi \;\Dashv\vDash\; \forall x\varphi \tag{16}$$

$$\forall x(\varphi \wedge \psi) \;\Dashv\vDash\; \varphi \wedge \forall x\psi \tag{17}$$

$$\forall x(\varphi \vee \psi) \;\Dashv\vDash\; \varphi \vee \forall x\psi \tag{18}$$

$$\exists x(\varphi \wedge \psi) \;\Dashv\vDash\; \varphi \wedge \exists x\psi \tag{19}$$

$$\exists x(\varphi \vee \psi) \;\Dashv\vDash\; \varphi \vee \exists x\psi \tag{20}$$

$$\forall x(\varphi \to \psi) \;\Dashv\vDash\; \varphi \to \forall x\psi \tag{21}$$

$$\exists x(\varphi \to \psi) \;\Dashv\vDash\; \varphi \to \exists x\psi \tag{22}$$

$$\forall x(\psi \to \varphi) \;\Dashv\vDash\; \exists x\psi \to \varphi \tag{23}$$

$$\exists x(\psi \to \varphi) \;\Dashv\vDash\; \forall x\psi \to \varphi \tag{24}$$

Definition 2.26

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

Notation: $Sent_{\mathcal{L}}$:= the set of sentences of $\mathcal{L}$.

Proposition 2.27

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

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

Definition 2.28

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

Let φ be a formula and Γ be a set of formulas of $\mathcal{L}$.

Definition 2.29

We say that Γ is satisfiable if there exists an $\mathcal{L}$-structure $\mathcal{A}$ and an $\mathcal{A}$-assignment e such that

$$\mathcal{A} \vDash \gamma[e] \text{ for all } \gamma \in \Gamma.$$

$(\mathcal{A}, e)$ is called a model of Γ.

Definition 2.30

We say that φ is a logical consequence of Γ if for all $\mathcal{L}$-structures $\mathcal{A}$ and all $\mathcal{A}$-assignments $e : V \to A$,

$$(\mathcal{A}, e) \text{ model of } \Gamma \implies (\mathcal{A}, e) \text{ model of } \varphi.$$

Notation: $\Gamma \vDash \varphi$

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

Definition 2.31

We say that Γ is satisfiable if there exists an $\mathcal{L}$-structure $\mathcal{A}$ such that

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

$\mathcal{A}$ is called a model of Γ. Notation: $\mathcal{A} \models \Gamma$

Definition 2.32

We say that 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: $\Gamma \models \varphi$

Tautologies

The notions of tautology and tautological consequence from propositional logic can also be applied to a first-order language $\mathcal{L}$. Intuitively, a tautology is a formula which is "true" based only on the interpretations of the connectives $\neg, \rightarrow$.

Definition 2.33

An $\mathcal{L}$-truth assignment is a function $F : Form_{\mathcal{L}} \rightarrow \{0, 1\}$ satisfying, for all formulas φ, ψ,

- $F(\neg\varphi) = 1 - F(\varphi)$;
- $F(\varphi \rightarrow \psi) = F(\varphi) \rightarrow F(\psi)$.

Definition 2.34

φ is a tautology if $F(\varphi) = 1$ for any $\mathcal{L}$-truth assignment F.

Examples of tautologies: $\varphi \rightarrow (\psi \rightarrow \varphi)$, $(\varphi \rightarrow \psi) \leftrightarrow (\neg\psi \rightarrow \neg\varphi)$

Proposition 2.35

If φ is a tautology, then φ is valid.

Example

$x = x$ is valid, but $x = x$ is not a tautology.

Definition 2.36

We say that the formulas φ and ψ are tautologically equivalent if $F(\varphi) = F(\psi)$ for any $\mathcal{L}$-truth assignment F.

Example 2.37

$\varphi_1 \to (\varphi_2 \to \ldots \to (\varphi_n \to \psi)\ldots)$ and $(\varphi_1 \wedge \ldots \wedge \varphi_n) \to \psi$ are tautologically equivalent.

Definition 2.38

Let φ be a formula and Γ be a set of formulas. We say that φ is a tautological consequence of Γ if for any $\mathcal{L}$-truth assignment F,

$$F(\gamma) = 1 \text{ for all } \gamma \in \Gamma \quad \Rightarrow \quad F(\varphi) = 1.$$

Proposition 2.39

If φ is a tautological consequence of Γ, then $\Gamma \vDash \varphi$.

Substitution

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

Definition 2.40

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

$t_x(u)$:= *the expression obtained from t by replacing all occurences of x with u.*

Proposition 2.41

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

Substitution

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

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

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.

Substitution

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

Definition 2.42

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 occurences 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 occurences of x.

Substitution

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

Definition 2.43

$$\varphi_x(u) \quad := \quad \textit{the expression obtained from } \varphi \textit{ by replacing all free occurences of } x \textit{ in } \varphi \textit{ with } u.$$

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

Proposition 2.44

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

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

Proposition 2.45

Let φ be a formula and x be a variable.

(i) *For any term u substitutable for x in φ,*

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

(ii) $\models \forall x\varphi \to \varphi$ *and* $\models \varphi \to \exists x\varphi$*.*

(iii) *For any constant symbol c,*

$$\models \forall x\varphi \to \varphi_x(c) \textit{ and} \models \varphi_x(c) \to \exists x\varphi.$$

Proposition 2.46

For any formula φ, distinct variables x and y such that $y \notin FV(\varphi)$ and y is substitutable for x in φ,

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

In particular, this holds if y is a new variable, that does not occur in φ.

We use Proposition 2.46 as follows: if $\varphi_x(u)$ is not a free substitution (that is x is not free for u in φ), then we replace φ with a logically equivalent formula φ' such that $\varphi'_x(u)$ is a free substitution .

Substitution

Definition 2.47

For any formula φ and any variables $y_1, \ldots, y_k$, the $y_1, \ldots, y_k$-free variant φ' of φ is inductively defined as follows:

- *if φ is an atomic formula, then φ' is φ;*
- *if $\varphi = \neg\psi$, then φ' is $\neg\psi'$;*
- *if $\varphi = \psi \to \chi$, then φ' is $\psi' \to \chi'$;*
- *if $\varphi = \forall z\psi$, then*

$$\varphi' = \begin{cases} \forall w \psi'_z(w) & \text{if } z \in \{y_1, \ldots, y_k\} \\ \forall z \psi' & \text{altfel;} \end{cases}$$

where w is the first variable in the sequence $v_0, v_1, \ldots,$ which does not occur in ψ' and is not among $y_1, \ldots, y_k$.

Definition 2.48

φ' is a variant of φ if it is the $y_1, \ldots, y_k$-free variant of φ for some variables $y_1, \ldots, y_k$.

Proposition 2.49

(i) For any formulas φ and φ', if φ' is a variant of φ, then $\varphi \models\!\!\!\!\!\!= \varphi'$;

(ii) For any formula φ and any term u, if the variables of u are among $y_1, \ldots, y_k$ and φ' is the $y_1, \ldots, y_k$-free variant of φ, then $\varphi'_x(u)$ is a free substitution.

Definition 2.50

The set $LogAx_{\mathcal{L}} \subseteq Form_{\mathcal{L}}$ of logical axioms of $\mathcal{L}$ consists of:

(i) *all tautologies.*

(ii) *formulas of the form*

$$t = t, \quad s = t \to t = s, \quad s = t \wedge t = u \to s = u,$$

for any terms s, t, u.

(iii) *formulas of the form*

$$t_1 = u_1 \wedge \ldots \wedge t_m = u_m \to ft_1 \ldots t_m = fu_1 \ldots u_m,$$

$$t_1 = u_1 \wedge \ldots \wedge t_m = u_m \to (Rt_1 \ldots t_m \leftrightarrow Ru_1 \ldots u_m),$$

for any $m \geq 1$, $f \in \mathcal{F}_m$, $R \in \mathcal{R}_m$ and any terms s_i, t_i $(i = 1, \ldots, m)$.

(iv) *formulas of the form*

$$\varphi_x(t) \to \exists x \varphi,$$

where $\varphi_x(t)$ is a free substitution ($\exists$-axioms).

Definition 2.51

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

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

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

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

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

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

Definition 2.52

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

(Γ0) *Every logical axiom is a* Γ*-theorem.*

(Γ1) *Every formula of* Γ *is a* Γ*-theorem.*

(Γ2) *If* φ *and* $\varphi \to \psi$ *are* Γ*-theorems, then* ψ *is a* Γ*-theorem.*

(Γ3) *If* $\varphi \to \psi$ *is a* Γ*-theorem and* $x \notin FV(\psi)$*, then* $\exists x\varphi \to \psi$ *is a* Γ*-theorem.*

(Γ4) *Only the formulas obtained by applying rules* (Γ0)*,* (Γ1)*,* (Γ2) *and* (Γ3) *are* Γ*-theorems.*

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

Notations

$$\Gamma \vdash_{\mathcal{L}} \varphi \quad := \quad \varphi \text{ is a } \Gamma\text{-theorem}$$

$$\vdash_{\mathcal{L}} \varphi \quad := \quad \emptyset \vdash_{\mathcal{L}} \varphi$$

Definition 2.53

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..

Definition 2.54

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

(i) θ_i *is an axiom;*

(ii) $\theta_i \in \Gamma$*;*

(iii) *there exist* $k, j < i$ *such that* $\theta_k = \theta_j \to \theta_i$*;*

(iv) *there exists* $j < i$ *such that*

$$\theta_j = \varphi \to \psi \text{ and } \theta_i = \exists x \varphi \to \psi,$$

where φ, ψ *are formulas and* $x \notin FV(\psi)$*.*

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

Definition 2.55

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

Proposition 2.56

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

$$\Gamma \vdash \varphi \quad \text{iff} \quad \text{there exists a } \Gamma\text{-proof of } \varphi.$$

Let Γ be a set of formulas.

Theorem 2.57 (Tautology Theorem (Post))

If ψ is a tautological consequence of $\{\varphi_1, \ldots, \varphi_n\}$ and $\Gamma \vdash \varphi_1, \ldots, \Gamma \vdash \varphi_n$, then $\Gamma \vdash \psi$.

Theorem 2.58 (Deduction Theorem)

Let $\Gamma \cup \{\psi\}$ be a set of formulas and φ be a sentence. Then

$$\Gamma \cup \{\varphi\} \vdash \psi \quad \text{iff} \quad \Gamma \vdash \varphi \to \psi.$$

Definition 2.59

Γ is called consistent if there exists a formula φ such that $\Gamma \nvdash \varphi$. Γ is said to be inconsistent if it is not consistent, that is $\Gamma \vdash \varphi$ for any formula φ.

Proposition 2.60

For any formula φ *and variable* x,

$$\Gamma \vdash \varphi \quad \Longleftrightarrow \quad \Gamma \vdash \forall x \varphi.$$

Definition 2.61

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

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

Notation 2.62

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

Proposition 2.63

For any formula φ,

$$\Gamma \vdash \varphi \quad \Longleftrightarrow \quad \Gamma \vdash \overline{\forall \varphi} \quad \Longleftrightarrow \quad \overline{\forall \Gamma} \vdash \varphi \quad \Longleftrightarrow \quad \overline{\forall \Gamma} \vdash \overline{\forall \varphi}.$$

Theorem 2.64 (Completeness Theorem (version 1))

Let Γ be a set of sentences. Then

$$\Gamma \text{ is consistent} \quad \Longleftrightarrow \quad \Gamma \text{ is satisfiable.}$$

Theorem 2.65 (Completeness Theorem (version 2))

For any set of sentences Γ and any sentence φ,

$$\Gamma \vdash \varphi \quad \Longleftrightarrow \quad \Gamma \models \varphi.$$

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

Prenex normal form

Definition 2.66

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

Definition 2.67

A formula φ is in prenex normal form if

$$\varphi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \, \psi,$$

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. $Q_1 x_1 Q_2 x_2 \dots Q_n x_n$ 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.

Prenex normal form

Examples of formulas in prenex normal form:

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

Theorem 2.68 (Prenex normal form theorem)

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

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

$$\begin{aligned} \varphi &\mathrel{\text{⫤}\!\vDash} \exists y\big(g(y,z) = c \wedge \neg\exists x(f(x) = d)\big) \\ &\mathrel{\text{⫤}\!\vDash} \exists y\big(g(y,z) = c \wedge \forall x\neg(f(x) = d)\big) \\ &\mathrel{\text{⫤}\!\vDash} \exists y\forall x\big(g(y,z) = c \wedge \neg(f(x) = d)\big) \end{aligned}$$

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

Example

Find a prenex normal form of the formula

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

$$\begin{aligned}
\varphi \;&\models\!\!\!\dashv\; \exists y \neg(S(y) \to \exists z R(z)) \wedge \forall x(\forall y P(x,y) \to f(x) = d) \\
&\models\!\!\!\dashv\; \exists y \neg\exists z(S(y) \to R(z)) \wedge \forall x(\forall y P(x,y) \to f(x) = d) \\
&\models\!\!\!\dashv\; \exists y \neg\exists z(S(y) \to R(z)) \wedge \forall x \exists y(P(x,y) \to f(x) = d) \\
&\models\!\!\!\dashv\; \exists y \forall z \neg(S(y) \to R(z)) \wedge \forall x \exists y(P(x,y) \to f(x) = d) \\
&\models\!\!\!\dashv\; \exists y \forall z \big(\neg(S(y) \to R(z)) \wedge \forall x \exists y(P(x,y) \to f(x) = d)\big) \\
&\models\!\!\!\dashv\; \exists y \forall z \forall x \big(\neg(S(y) \to R(z)) \wedge \exists y(P(x,y) \to f(x) = d)\big) \\
&\models\!\!\!\dashv\; \exists y \forall z \forall x \big(\neg(S(y) \to R(z)) \wedge \exists v(P(x,v) \to f(x) = d)\big) \\
&\models\!\!\!\dashv\; \exists y \forall z \forall x \exists v \big(\neg(S(y) \to R(z)) \wedge (P(x,v) \to f(x) = d)\big)
\end{aligned}$$

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

Logic for Multiagent Systems

Master 1st Year, 1st Semester 201/2022

Laurențiu Leuștean

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

Multiagent systems

Credits and ackowledgements

The lecture follows

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

[2] Lecture slides/handouts, made available by Michael Wooldridge here

[3] Yoav Shoham, Kevin Leyton-Brown, Multiagents Systems, Cambridge University Press, 2009

- Most of the material on the slides is taken from [1,2]. Some material is taken from [3].
- The figures are taken from [1,2].

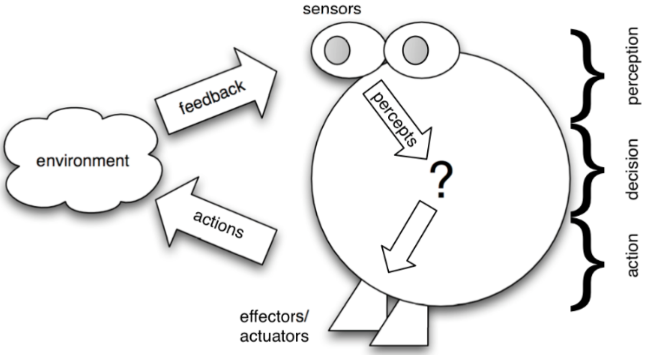
Agents

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

Definition in [1,2]

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

Agents



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

Multiagent systems

Definition in [1,2]

A multiagent system is one that consists of a number of agents, which interact with one-another.
Agents act on behalf of users with different goals and motivations.
To successfully interact, they require the ability to cooperate, coordinate, and negotiate with each other, much as people do.

Definition in [3]

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

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

Multiagent systems

Examples

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

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

Intelligent agents

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

Reactivity

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

Proactiveness

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

Social ability

- The ability to interact with other agents (and possibly humans) via cooperation, coordination, and negotiation.
- Cooperation is working together as a team to achieve a shared goal. It gives a better result.
- Coordination is managing the interdependencies between activities. For example, if there is a non-sharable resource that you want to use and I want to use, then we need to coordinate.
- Negotiation is the ability to reach agreements on matters of common interest. Typically involves offer and counter-offer, with compromises made by participants.

Abstract architectures for intelligent agents

Make formal the abstract view of agents.

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

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

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

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

- 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, ...
- actions are denotes also by α_0, α_1, ...,

Abstract architectures for intelligent agents

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

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

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

Definition 1.1

A run r over E and Ac is a finite sequence

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

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 $r, r', \ldots$. We write a run r as follows:

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

or

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

Notation 1.2

- $\mathcal{R}$ denotes the set of all runs (over E and 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.

Definition 1.3

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

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

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

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

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

Definition 1.4

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

Abstract architectures for intelligent agents

We introduce a model of the agents that inhabit systems.

Definition 1.5

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

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

Definition 1.6

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

Definition 1.7

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

- $x_0 = e_0$.
- *for all* $k \geq 0$:

$$x_{2k+1} = Ag(x_0, \ldots, x_{2k}) \text{ and } x_{2k+2} \in \tau(x_0, \ldots, x_{2k+1}).$$

- *r is terminated in the following sense:*
  - *if* $x_n \in E$, *then* $\tau(r) = \emptyset$;
  - *if* $x_n \in Ac$, *then* $Ag(r) = null$.

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

Notation 1.8

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

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 1.9

Two agents Ag_1 and Ag_2 are said to be

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

Purely reactive agents

- Certain types of agents decide what to do without reference to their history. They base their decision-making entirely on the present, with no reference at all to the past.
- We call such agents purely reactive, since they simply respond directly to their environment.

Definition 1.10

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

Example: Thermostat

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

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

$$\begin{array}{lcl} \text{temperature is too low} & \to & \text{heating on,} \\ \text{temperature is OK} & \to & \text{heating off.} \end{array}$$

Example: Thermostat

- Let us denote

  $e_1 :=$ temperature too low, $\quad e_2 :=$ temperature OK,

  $\alpha_1 :=$ heating on, $\quad \alpha_2 :=$ 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:

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

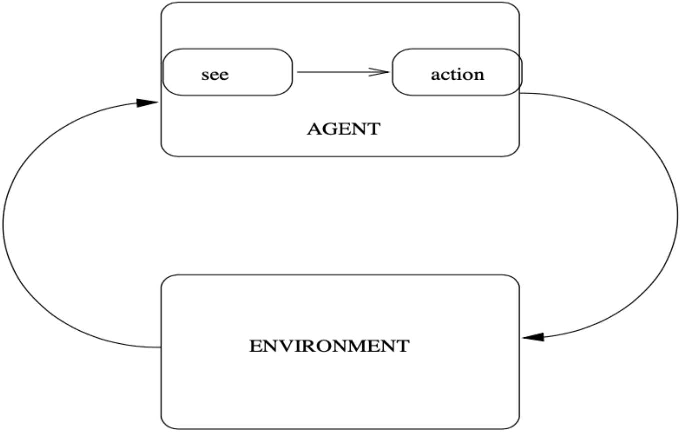
Intelligent agents

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

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

Perception

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



Perception

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

Let Per be a nonempty set of percepts.

Definition 1.11

The see and action functions are defined as follows:

$$see : E \to Per \qquad \text{and} \qquad action : Per^* \to Ac.$$

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

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

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 $see(e_1) = see(e_2)$. Then e_1 and e_2 are mapped to the same percept, and the agent receives the same perceptual information from each of them. As far as the agent is concerned, e_1 and e_2 are indistinguishable.

Definition 1.12

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

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

Remark 1.13

$\equiv$ is an equivalence relation on E.

Perception

- $\equiv$ partitions E into mutually indistinguishable sets of states, namely the different equivalence classes $[e]$, were $e \in E$.

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

Example

- Let us consider the statements

$$x \quad := \quad \text{“the room temperature is OK”}$$
$$y \quad := \quad \text{“Angela Merkel is the German Chancellor”}$$

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

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

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

Perception - an example

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

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

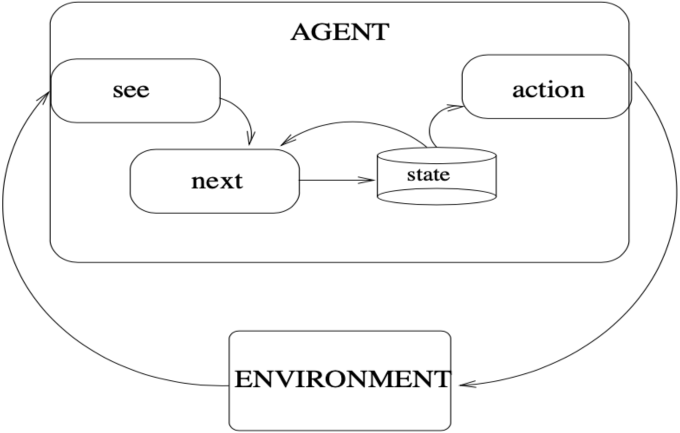
The perception function *see* is defined as follows:

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

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

Agents with state

We now consider agents that maintain state.



Agents with state

- These agents have some internal data structure, which is typically used to record information about the environment state and history. Let I be the set of all internal states of the agent.

Definition 1.14

The see and action functions are defined as follows:

$$see : E \to Per \qquad and \qquad action : I \to Ac.$$

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

Definition 1.15

The function next is defined as follows:

$$next : I \times Per \to I.$$

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 $see(e)$.
- The internal state of the agent is then updated to $i_1 := next(i_0, see(e))$.
- The action selected by the agent is $\alpha := action(i_1)$.
- The agents 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*.

Tasks for agents

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

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

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

Tasks for agents

- We want to tell our agent what to do without telling it how to do it.
- One way of doing this is to define tasks indirectly, via some kind of performance measure.
- One possibility: associate utilities with states of the environment.

- A utility is a numeric value representing how ‘good’ a state is: the higher the utility, the better.
- The task of the agent is then to bring about states that maximize utility.
- We do not specify to the agent how this is to be done.

Utility functions

Definition 1.16

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

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.

Utility functions

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

Definition 1.17

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

- If we are concerned with agents that must operate independently over long periods of time, then this approach is appropriate.

- The utility-based approach works well in certain scenarios.
- Problems:
  - Sometimes it is difficult to define a utility function.
  - People don't think in terms of utilities. It is hard for people to specify tasks in these terms.

Tileworld

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

- Simulated two dimensional grid environment on which there are agents, tiles, obstacles, and holes.
- An agent can move in four directions, up, down, left, or right, and if it is located next to a tile, it can push it.
- An obstacle is a group of immovable grid cells.
- Holes have to be filled up with tiles by the agent.
- An agent scores points by filling holes with tiles, the aim being to fill as many holes as possible.
- Holes appear randomly and exist for as long as their life expectancy, unless they disappear because of the agent's actions.
- The interval between the appearance of successive holes is called the hole gestation time.

Tileworld

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

Definition 1.18

The utility function *is defined as follows:*

$$u : \mathcal{R} \to \mathbb{R}, \qquad u(r) = \frac{\textit{number of holes filled in } r}{\textit{number of holes that appeared in } r}$$

- 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

Tileworld

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

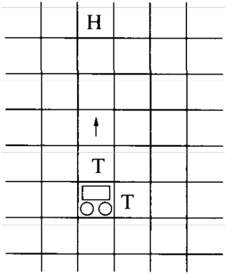


Figure 1: Tileworld example

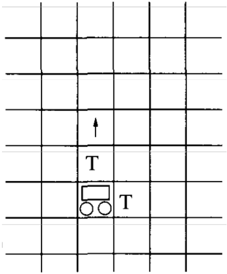


Figure 2: Tileworld example

Example 1.19

Suppose an agent is pushing a tile to a hole (Figure 1), when this hole disappears (Figure 2).
The agent should recognize this change in the environment, and modify its behaviour appropriately.

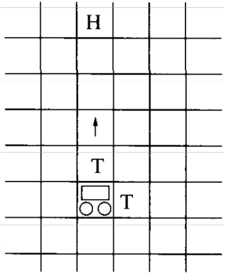


Figure 3: Tileworld example

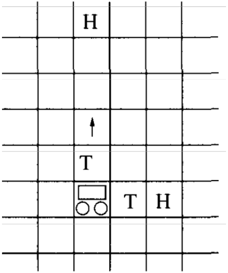


Figure 4: Tileworld example

Example 1.20

Suppose an agent is pushing a tile to a hole (Figure 3), when a hole appears to the right of the agent (Figure 4).
It would do better to push the tile to the right, than to continue to head north, for the simple reason that it only has to push the tile one step, rather than three.

Expected utility

Notation 1.22

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

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

Definition 1.23

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

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

Expected utility - an example

Consider the environment (E, e_0, 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

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

We consider the following two agents in this environment:

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

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

Runs of Ag_1 and Ag_2 in the environment Env

$$\mathcal{R}(Ag_1, Env) = \{r_1 := e_0 \xrightarrow{\alpha_0} e_1, r_2 := e_0 \xrightarrow{\alpha_0} e_2\},$$
$$\mathcal{R}(Ag_2, Env) = \{r_3 := e_0 \xrightarrow{\alpha_1} e_3, r_4 := e_0 \xrightarrow{\alpha_1} e_4, r_5 := e_0 \xrightarrow{\alpha_1} e_5\}.$$

The probabilities of the various runs are defined as follows:

- for the agent Ag_1:

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

- for the agent Ag_2:

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

Expected utility - an example

Assume the utility function u is defined as follows:

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

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

$$\begin{aligned}
EU(Ag_1, Env) &= u(r_1)P(r_1 \mid Ag_1, Env) + u(r_2)P(r_2 \mid Ag_1, Env) \\
&= 8 \times 0.4 + 11 \times 0.6 = 9.6 \\
EU(Ag_2, Env) &= u(r_3)P(r_3 \mid Ag_2, Env) + u(r_4)P(r_4 \mid Ag_2, Env) \\
&\quad + u(r_5)P(r_5 \mid Ag_2, Env) \\
&= 70 \times 0.1 + 9 \times 0.2 + 10 \times 0.7 = 15.7
\end{aligned}$$

Optimal agents

Notation 1.24

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

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

Definition 1.25

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

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

- The fact that an agent is optimal does not mean that it will be best; only that on average, we can expect it to do best.
- The definition does not not give us any clues about how to implement this agent.
- There are agents that cannot be implemented on a real computer.

Bounded optimal agents

Suppose m is a particular computer.

Notation 1.26

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

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

Definition 1.27

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

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

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

Predicate task specifications

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

Definition 1.28

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

Task environment

Definition 1.29

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

Notation 1.30

TE denotes the set of all task environments.

A task environment specifies:

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

Notation 1.31

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

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 1.32

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

Achievement and maintenance tasks

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

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

- Achievement tasks are those of the form 'achieve state of affairs φ'.
- Maintenance tasks are those of the form 'maintain state of affairs φ'.

Achievement tasks

Definition 1.33

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 1.34

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

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

Achievement tasks

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

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

Maintenance tasks

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

Definition 1.35

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

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

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

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

Notation 1.36

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

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.

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.

Agent synthesis

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

There is at least one other possibility:

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

Agent synthesis

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

Definition 1.37

An agent synthesis algorithm is a function

$$syn : TE \to AG \cup \{\perp\}.$$

A synthesis algorithm is

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

Definition 1.38

An agent synthesis algorithm syn is

- *sound if for any task environment* $(Env, \Psi) \in TE$,

  $syn((Env, \Psi)) = Ag$ *implies* $\mathcal{R}(Ag, Env) = \mathcal{R}_\Psi(Ag, Env)$.
- *complete if for any* $(Env, \Psi) \in TE$,

  $\big($*there exists an agent* Ag *s.t.* $\mathcal{R}(Ag, Env) = \mathcal{R}_\Psi(Ag, Env)\big)$

  implies $syn((Env, \Psi)) \neq \perp$.

- 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.

Agent synthesis

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

Agent architectures

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.

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.

Logic-based agents

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

Key ideas:

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

Problems to be solved:

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

Agents as theorem provers

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

- An internal state of such an agent is a database of formulas of classical first-order logic.
- The agent's database might contain formulas such as

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

Agents as theorem provers

We use the model of agents with state.
Let $\mathcal{L}$ be a first-order language and $Form_{\mathcal{L}}$ be the set of its formulas.

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

- a unary relation symbol Do;
- a constant symbol c_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 $DB, DB_1, \ldots$ for members of $\mathcal{D}$.
- An internal state of the agent is a database. Thus, $I = \mathcal{D}$.

Agents as theorem provers

- We fix a set $\Sigma \subseteq Form_{\mathcal{L}}$ 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

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

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

Agents as theorem provers

```
function action(DB : 𝒟) returns an action Ac
begin
   for each α ∈ Ac do
      if DB ⊢_Σ Do(α) then
         return α
      end-if
   end-for
   for each α ∈ Ac do
      if DB ⊬_Σ ¬Do(α) then
         return α
      end-if
   end-for
   return null
end function action
```

Figure 5: Agent selection as theorem proving.

Agents as theorem provers

- In lines 3-7, the agent takes each of its possible actions 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.

Agents as theorem provers

The perception function *see* remains unchanged:

$$see : E \to Per,$$

where *Per* is the set of percepts.

The *next* function has the form:

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

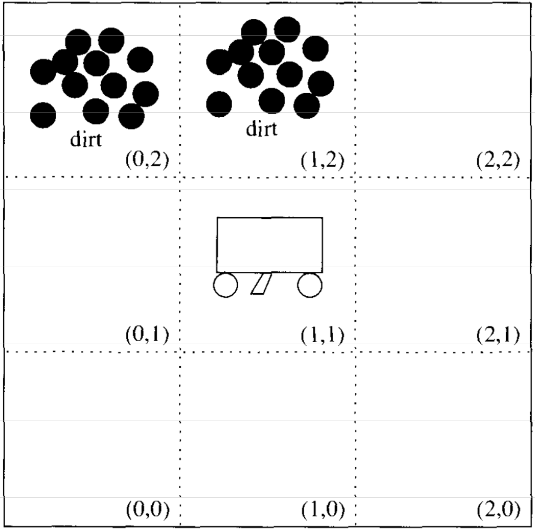
It maps a database and a percept to a new database.

Agents as theorem provers - example

We consider an example: vacuum cleaning world

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

Agents as theorem provers - example



Agents as theorem provers - example

- The set *Per* of percepts is defined as
$$Per = \{dirt, nothing\},$$
where *dirt* signifies that there is dirt beneath it, and *nothing* indicates no special information.
- The set *Ac* of actions is defined as
$$Ac = \{forward, suck, turn\},$$
where *forward* means 'go forward', *suck* means 'suck up dirt', and *turn* means 'turn right 90 degrees'.
- The goal is to traverse the room continually searching for and removing dirt.

Agents as theorem provers - example

We use three simple domain predicates:

$In(i,j)$	agent is at (i,j),
$Dirt(i,j)$	there is dirt at (i,j),
$Facing(d)$	the agent is facing direction d,

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

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

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

Agents as theorem provers - example

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

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

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

where

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

Agents as theorem provers - example

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

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

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

The next function is defined as follows:

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

Agents as theorem provers - example

The deduction formulas have the general form

$$\varphi \to \psi, \qquad \text{where } \varphi,\ \psi \text{ are formulas of } \mathcal{L}$$

Cleaning

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

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

Agents as theorem provers - example

Traversal

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

$$
\begin{array}{rcl}
In(0,0) \land Facing(north) \land \neg Dirt(0,0) & \rightarrow & Do(forward) \\
In(0,1) \land Facing(north) \land \neg Dirt(0,1) & \rightarrow & Do(forward) \\
In(0,2) \land Facing(north) \land \neg Dirt(0,2) & \rightarrow & Do(turn) \\
In(0,2) \land Facing(east) & \rightarrow & Do(forward)
\end{array}
$$

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

Logic-based agents

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

Advantages:

- elegance and a clean (logical) semantics.

Disadvantages:

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

Multiagent interactions

Multiagent interactions

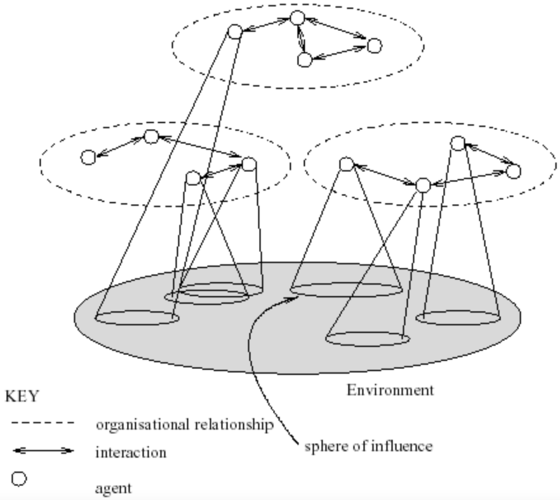


Figure 6: Typical structure of a multiagent system

Multiagent interactions

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

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

Multiagent interactions

We denote the set of agents by AG. Then

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

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

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

Utilities and preferences

Definition 1.39

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

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

Definition 1.40

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

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

- If $\omega \succeq_i \omega'$, we say that agent i prefers ω over ω' or that ω is prefered by agent i over ω'.

Definition 1.41

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

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

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

Proposition 1.42

- *$\succeq_i$ is reflexive, transitive and total.*
- *$\succ_i$ is ireflexive and transitive.*

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

Example 1.43

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

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

Multiagent interactions

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

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

Notation 1.44

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

Definition 1.45

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

Definition 1.46

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

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

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

$$AG = \{1, 2\}, \qquad Ac_1 = Ac_2 = \{C, D\} \text{ and}$$
$$Ac = \{(D, D), (D, C), (C, D), (C, C)\}.$$

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

Example 1.47

Define

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

τ maps each action profile to a different outcome. The environment is sensitive to the actions of both agents.

Example 1.48

Define

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

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

Example 1.49

Define

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

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

Multiagent interactions - Example

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

$$u_1(\omega_1) = 1, \quad u_1(\omega_2) = 1, \quad u_1(\omega_3) = 4, \quad u_1(\omega_4) = 4,$$
$$u_2(\omega_1) = 1, \quad u_2(\omega_2) = 4, \quad u_2(\omega_3) = 1, \quad u_2(\omega_4) = 4.$$

- With a bit of abuse of notation, we write

$$u_1(D, D) = 1, \quad u_1(D, C) = 1, \quad u_1(C, D) = 4, \quad u_1(C, C) = 4,$$
$$u_2(D, D) = 1, \quad u_2(D, C) = 4, \quad u_2(C, D) = 1, \quad u_2(C, C) = 4.$$

Multiagent interactions - Example

- Agent 1's preferences are:

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

- Consider the following question:

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

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

- Agent 2's preferences are:

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

- The answer is again clear: agent 2 should cooperate; C is the rational choice for agent 2.

- In this scenario, the action one agent should perform does not depend in any way on what the other does.
- If both agents act rationally, then the joint action selected will be (C, C): both agents will cooperate.

Multiagent interactions - Example

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

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

- The two agents' preferences are:

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

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

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

Pay-off matrix

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

		1	
		D	C
	D	4	1
2		4	4
	C	4	1
		1	1

Corresponds to

$$u_1(D, D) = 4, \quad u_1(D, C) = 4, \quad u_1(C, D) = 1, \quad u_1(C, C) = 1,$$
$$u_2(D, D) = 4, \quad u_2(D, C) = 1, \quad u_2(C, D) = 4, \quad u_2(C, C) = 1.$$

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

The basic question for each agent is:

What should I do?

We define some solution concepts and solution properties.

Solution Concepts and Solution Properties

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

We use terminology from game theory.

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

Definition 1.50

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

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

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

Definition 1.51

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

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

Dominant strategies

Similar definitions are given for agent 2.

Definition 1.52

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

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

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

Definition 1.53

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

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

Dominant strategies

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

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

Recall the example with the following pay-off matrix:

		1			
		D		C	
	D		4		1
2		4		4	
	C		4		1
		1		1	

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

Definition 1.54

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

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

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

Remark 1.55

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

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

A reformulation

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

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

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

Remark 1.56

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

Nash equilibrium

- Nash equilibrium was introduced by John Forbes Nash, Jr. in his 1950 PhD thesis.
- One of the most important concepts in game theory.
- One of the most important concepts in analysing multiagent systems.

- The type of Nash equilibrium defined above is known as pure strategy Nash equilibrium.
- If in a given scenario there exists a Nash equilibrium, then it is clear what to do.

Unfortunately,

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

Matching Pennies

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

Pay-off matrix

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

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

Rock-Paper-Scissors

This game is played by two players, as follows.

- The players must simultaneously declare one of three possible choices: rock, scissors, or paper.
- They usually do this by showing a fist for rock, an open palm for paper, or a V-shape with the fingers for scissors.
- To determine the winner, the following rule is used:

  Paper covers rock; scissors cut paper; rock blunts scissors.

- In other words, if you declare paper and I declare rock, then you win; if you declare paper and I declare scissors, then I win; if I declare scissors and you declare rock, then you win, and so on.

Rock-Paper-Scissors

Pay-off matrix

		1		
		rock	paper	scissors
	rock	0 0	1 -1	-1 1
2	paper	-1 1	0 0	1 -1
	scissors	1 -1	-1 1	0 0

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

Definition 1.57

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

Remark 1.58

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

Let ω be an outcome.

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

Pareto efficiency

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

Example 1.59

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

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

Social welfare

Definition 1.60

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

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

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

Social welfare

Maximizing social welfare

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

Example 1.61

Suppose you have a game with 100 players, in which one outcome gives 101 million to one player and nothing to the rest, while every other outcome gives 1 million to each.
The first outcome maximizes social welfare, although the other 99 players might not be very happy with such an outcome.

Definition 1.62

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

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

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

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

Definition 1.63

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

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

There is no no possibility of cooperative behaviour:

- The best outcome for you is the worst outcome for your opponent.
- If you allow your opponent to get positive utility, then you get negative utility.

- Matching Pennies and Rock-Paper-Scissors are zero-sum games.

Prisoner's Dilemma

The following is a famous game-theoretic scenario.

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

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

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

Prisoner's Dilemma

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

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

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

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

Prisoner's Dilemma

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

		1	
		D	C
2	D	3 3	2 5
	C	5 2	4 4

Thus,

$$u_1(D,D)=3, \quad u_1(D,C)=5, \quad u_1(C,D)=2, \quad u_1(C,C)=4,$$
$$u_2(D,D)=3, \quad u_2(D,C)=2, \quad u_2(C,D)=5, \quad u_2(C,C)=4.$$

Prisoner's Dilemma

The two players' preferences are:

$$(D, C) \succ_1 (C, C) \succ_1 (D, D) \succ_1 (C, D),$$
$$(C, D) \succ_2 (C, C) \succ_2 (D, D) \succ_2 (D, C).$$

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

Prisoner's Dilemma

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

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

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

Example 1.64

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

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

Logical foundations

Logics of knowledge and belief

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

We look at how one might represent statements such as

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

Logics of knowledge and belief

Consider the following statement:

Janine believes Cronos is the father of Zeus.

How can we formalize it it in first-order logic?

An attempt is

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

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

A syntactic problem

φ is not a formula of first-order logic, as $\mathit{Father}(\mathit{Zeus}, \mathit{Cronos})$ is a formula, not a term.

A semantic problem

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

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

- Let us denote

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

- If φ, χ were formulas of first-order logic, then, from the truth of ψ, we could infer that φ and χ are equivalent.
- Intuition rejects this derivation as invalid: believing that the father of Zeus is Cronos is not the same as believing that the father of Jupiter is Cronos.

A semantic problem

The problem is that, in general the truth value of the sentence

Janine believes p.

does not dependent solely on the truth value of p. In other words, belief is not truth-functional.
Similarly, knowledge is not truth-functional.

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

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

Logics of knowledge and belief

Approach to the syntactic problem:

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

Approach to the semantic problem:

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

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

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

Modal Logics

Textbook:

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

Definition 1.65

The basic modal language ML_0 consists of:

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

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

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

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

Definition 1.66

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

(F0) *Every atomic proposition is a formula.*

(F1) $\perp$ *is a formula.*

(F2) *If* φ *is a formula, then* $(\neg\varphi)$ *is a formula.*

(F3) *If* φ *and* ψ *are formulas, then* $(\varphi \to \psi)$ *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 $Form(ML_0)$.

Basic modal language

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

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

Derived connectives

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

$$\begin{array}{ll} \varphi \vee \psi := ((\neg\varphi) \to \psi) & \varphi \wedge \psi := \neg(\varphi \to (\neg\psi)) \\ \varphi \leftrightarrow \psi := ((\varphi \to \psi) \wedge (\psi \to \varphi)) & \top := \neg\bot. \end{array}$$

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 $\neg\varphi, \varphi \to \psi$, $\Box\varphi$.

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 $\to, \wedge, \vee, \leftrightarrow$;
- $\wedge, \vee$ have higher precedence than $\to, \leftrightarrow$.

Classical modal logic

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

Examples of formulas we would probably regard as correct principles

- $\Box\varphi \to \Diamond\varphi$ (whatever is necessary is possible)
- $\varphi \to \Diamond\varphi$ (whatever is, is possible).

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

Definition 1.67

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

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

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

Example 1.68

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

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

Definition 1.69

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

$$R_a \subseteq W \times W$$

is a binary relation on W.

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

Relational structures

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:

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

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

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

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

Frames and models

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

We will do this in two distinct ways:

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

Definition 1.70

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

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

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

Frames and models

Definition 1.71

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

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

Thus, V assigns to each atomic proposition $p \in PROP$ a subset $V(p)$ of W. Informally, we think of $V(p)$ as the set of points in the model $\mathcal{M}$ where p is true.
Note that models for ML_0 can also be viewed as relational structures in a natural way:

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

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

Frames and models

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

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

Remark

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

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

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

Frames and models

We define now the notion of satisfaction.

Definition 1.72

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

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

$$
\begin{array}{rll}
\mathcal{M}, w \Vdash p & \text{iff} & w \in V(p), \quad \text{where } p \in PROP \\
\mathcal{M}, w \Vdash \bot & & \text{never} \\
\mathcal{M}, w \Vdash \neg\varphi & \text{iff} & \text{it is not true that } \mathcal{M}, w \Vdash \varphi \\
\mathcal{M}, w \Vdash \varphi \to \psi & \text{iff} & \mathcal{M}, w \Vdash \varphi \text{ implies } \mathcal{M}, w \Vdash \psi \\
\mathcal{M}, w \Vdash \Box\varphi & \text{iff} & \text{for every } v \in W, Rwv \text{ implies } \mathcal{M}, v \Vdash \varphi.
\end{array}
$$

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

Notation

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

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

- $\mathcal{M}, w \nVdash \perp$
- $\mathcal{M}, w \Vdash \neg\varphi$ iff $\mathcal{M}, w \nVdash \varphi$.

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:

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

Frames and models

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

Proposition 1.73

For every formulas φ, ψ,

$$\begin{array}{lll} \mathcal{M}, w \Vdash \varphi \vee \psi & \textit{iff} & \mathcal{M}, w \Vdash \varphi \textit{ or } \mathcal{M}, w \Vdash \psi \\ \mathcal{M}, w \Vdash \varphi \wedge \psi & \textit{iff} & \mathcal{M}, w \Vdash \varphi \textit{ and } \mathcal{M}, w \Vdash \psi \end{array}$$

Proposition 1.74

For every formula φ,

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

Frames and models

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

Proposition 1.75

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

$$\Diamond^n \varphi := \underbrace{\Diamond\Diamond \ldots \Diamond}_{n\ times} \varphi, \qquad \Box^n \varphi := \underbrace{\Box\Box \ldots \Box}_{n\ times} \varphi.$$

Then

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

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

Definition 1.76

- *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 1.77

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 $\mathcal{M}, w \Vdash \Sigma$.*

Frames and models

Example 1.78

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

$$w_1 \longrightarrow w_2 \longrightarrow w_3 \longrightarrow w_4 \longrightarrow w_5$$

Let us choose a valuation V such that $V(p) = \{w_2, w_3\}$, $V(q) = \{w_1, w_2, w_3, w_4, w_5\}$ and $V(r) = \emptyset$.
Consider the model $\mathcal{M} = (\mathcal{F}, V)$. Then

(i) $\mathcal{M}, w_1 \Vdash \Diamond\Box p$

(ii) $\mathcal{M}, w_1 \nVdash \Diamond\Box p \to p$

(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) $\mathcal{M} \Vdash \Box q$.

Frames and models

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

Frames and models

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?

Frames and validity

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

Definition 1.79

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.

Frames and validity

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

Example 1.80

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

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

Frames and validity

Definition 1.81

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 1.82

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

Example 1.83

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

Example 1.84

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

Definition 1.85

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

Example 1.86

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

Modal consequence

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 $\boldsymbol{S}$.

Modal consequence

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

Definition 1.87

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

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

Notation: $\Sigma \Vdash_{\boldsymbol{S}} \varphi$

Thus, if Σ is true at a state of the model, then φ must be true at the same state.

Remark 1.88

$$\{\psi\} \Vdash_{\boldsymbol{S}} \varphi \textit{ iff } \boldsymbol{S} \Vdash \psi \rightarrow \varphi.$$

Example 1.89

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

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

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

Definition 1.90

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

- *Λ contains the following axioms:*

$$
\begin{array}{rl}
(Taut) & \text{all propositional tautologies}, \\
(K) & \Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi),
\end{array}
$$

  where φ, ψ are formulas of ML_0.

- *Λ is closed under the following deduction rules:*
  - *modus ponens (MP):* $\dfrac{\varphi,\ \varphi \to \psi}{\psi}$.
    Hence, if $\varphi \in \Lambda$ and $\varphi \to \psi \in \Lambda$, then $\psi \in \Lambda$.
  - *generalization or necessitation:* $\dfrac{\varphi}{\Box\varphi}$.
    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,

$$\begin{array}{ll} (A1) & \varphi \to (\psi \to \varphi) \\ (A2) & (\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)) \\ (A3) & (\neg\psi \to \neg\varphi) \to (\varphi \to \psi). \end{array}$$

Proposition 1.91

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

Remark 1.92

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

Axiom (K) is sometimes called the distribution axiom and it is important because it allows us to transform $\Box(\varphi \to \psi)$ (a boxed formula) in an implication $\Box\varphi \to \Box\psi$, 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 $\Box(\varphi \to \psi)$ and $\Box\varphi$. Applying (K) and modus ponens, we get $\Box\varphi \to \Box\psi$. Applying again modus ponens, we obtain $\Box\psi$.

By Example 1.83,

Proposition 1.93

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

Normal modal logics

Theorem 1.94

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

Lemma 1.95

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

Definition 1.96

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

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

$\boldsymbol{K}$

Definition 1.97

*A **K**-proof is a sequence of formulas $\theta_1, \ldots, \theta_n$ such that for any $i \in \{1, \ldots, n\}$, 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 1.98

*Let φ be a formula. A **K**-proof of φ is a **K**-proof $\theta_1, \ldots, \theta_n$ such that $\theta_n = \varphi$.*
*If φ has a **K**-proof, we say that φ is **K**-provable.*
Notation: $\vdash_{\boldsymbol{K}} \varphi$.

Theorem 1.99

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

The logic $\boldsymbol{K}$ is very weak. If we are interested in transitive frames, we would like a proof system which reflects this. For example, we know that $\Box\varphi \to \Box\Box\varphi$ is valid in the class of all transitive frames, so we would want a proof system that generates this formula. $\boldsymbol{K}$ does not do this, since $\Box\varphi \to \Box\Box\varphi$ is not valid in the class of all frames.

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

$K\Gamma$

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

Definition 1.100

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

Definition 1.101

*A **K**Γ-proof is a sequence of formulas $\theta_1, \ldots, \theta_n$ such that for any $i \in \{1, \ldots, n\}$, 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 1.102

Let φ be a formula. A $\boldsymbol{K}\Gamma$-proof of φ is a $\boldsymbol{K}\Gamma$-proof $\theta_1, \ldots, \theta_n$ such that $\theta_n = \varphi$.
If φ has a $\boldsymbol{K}\Gamma$-proof, we say that φ is $\boldsymbol{K}\Gamma$-provable.
Notation: $\vdash_{\boldsymbol{K}\Gamma} \varphi$.

Theorem 1.103

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

Let Λ be a normal modal logic.

Definition 1.104

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 $\not\vdash_\Lambda \varphi$.

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 $\vdash_\Lambda \varphi \to \psi$, then $\vdash_\Lambda \psi$.

(iv) If $\vdash_\Lambda \varphi$, then $\vdash_\Lambda \Box\varphi$.

Definition 1.105

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

$$(\psi_1 \wedge \ldots \wedge \psi_n) \to \varphi \text{ is a tautology.}$$

Proposition 1.106

Λ is closed under propositional deduction: if φ is deducible in propositional logic from assumptions $\psi_1, \ldots, \psi_n$, then

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

Definition 1.107

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 $\psi_1, \ldots, \psi_n \in \Gamma$ $(n \geq 0)$ such that

$$\vdash_\Lambda (\psi_1 \wedge \ldots \wedge \psi_n) \to \varphi.$$

(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.

Proposition 1.108 (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* $\Delta \vdash_\Lambda \varphi$.

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

Proposition 1.109

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

Proposition 1.110

(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 $\Gamma \vdash_{\Lambda} \varphi \to \psi$, then $\Gamma \vdash_{\Lambda} \psi$.*

(iii) *If $\Gamma \vdash_{\Lambda} \varphi$ and $\{\varphi\} \vdash_{\Lambda} \psi$, then $\Gamma \vdash_{\Lambda} \psi$.*

Proposition 1.111 (Deduction Theorem)

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

$$\Gamma \vdash_{\Lambda} \varphi \to \psi \quad \textit{iff} \quad \Gamma \cup \{\varphi\} \vdash_{\Lambda} \psi.$$

Definition 1.112

A set Γ *of formulas is called* **Λ-consistent** *if* $\Gamma \not\vdash_\Lambda \bot$.
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 1.113

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) $\Gamma \vdash_\Lambda \varphi$ *for any formula* φ.

Proposition 1.114

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

Notation:

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

Definition 1.115

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

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

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

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

Definition 1.116

A normal logic Λ *is*

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

$$\Gamma \Vdash_{\boldsymbol{S}} \varphi \quad \textit{implies} \quad \Gamma \vdash_{\Lambda} \varphi.$$

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

$$\boldsymbol{S} \Vdash \varphi \quad \textit{implies} \quad \vdash_{\Lambda} \varphi.$$

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

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

Remark 1.117

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

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

Given a semantically specified normal logic $\Lambda_{\boldsymbol{S}}$ (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 $\Lambda_{\boldsymbol{S}}$ is the logic generated by Γ; we say that Γ axiomatizes $\boldsymbol{S}$.

*Completeness theorem for **K***

Theorem 1.118

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

*Logic **K**4*

Let

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

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

Theorem 1.119

***K**4 is sound and strongly complete with respect to the class of transitive frames.*

Logic T

Let

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

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

Definition 1.120

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

Theorem 1.121

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

Logic B

Let

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

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

Definition 1.122

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

Theorem 1.123

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

Logic KD

Let

$$\begin{array}{ll} (D) & \Box\varphi \to \Diamond\varphi \\ (D') & \neg\Box(\varphi \wedge \neg\varphi) \end{array}$$

One can easily see that $\vdash_{\boldsymbol{K}} (D) \leftrightarrow (D')$.

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

Definition 1.124

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

Theorem 1.125

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

Logic $\boldsymbol{K}5$

Let

$$\begin{array}{ll} (5) & \Diamond\varphi \to \Box\Diamond\varphi \\ (5') & \neg\Box\varphi \to \Box\neg\Box\varphi \end{array}$$

One can easily see that $\vdash_{\boldsymbol{K}} (5) \leftrightarrow (5')$.

Let $\boldsymbol{K}5$ be the normal logic generated by (5) (or, equivalently, by (5′)).

Definition 1.126

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

$$\textit{if } Rwv \textit{ and } Rwu\textit{, then } Rvu.$$

Theorem 1.127

$\boldsymbol{K}5$ is sound and strongly complete with respect to the class of Euclidean frames.

Logic $\boldsymbol{S}4$

We use the notation $\boldsymbol{S}4$ for the normal logic $\boldsymbol{KT}4$ generated by (T) and (4).

Theorem 1.128

$\boldsymbol{S}4$ *is sound and strongly complete with respect to the class of reflexive and transitive frames.*

Logic $\boldsymbol{S}5$

We use the notation $\boldsymbol{S}5$ for the normal logic $\boldsymbol{KT}4\boldsymbol{B}$ generated by (T), (4) and (B).

Proposition 1.129

$\boldsymbol{S}5 = \boldsymbol{KDB}4 = \boldsymbol{KDB}5 = \boldsymbol{KT}5.$

Theorem 1.130

$\boldsymbol{S}5$ is sound and strongly complete with respect to the class of frames whose relation is an equivalence relation.

Multimodal logics

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

Let I be a nonempty set.

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

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

  where $p \in PROP$ and $i \in I$.
- The dual of $\Box_i$ is denoted by $\Diamond_i$ and is defined as:

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

Multimodal logics

- 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 $V : PROP \to 2^W$ is a valuation.
- The last clause from the definition of the satisfaction relation $\mathcal{M}, w \Vdash \varphi$ is changed to: for all $i \in I$,

  $$\mathcal{M}, w \Vdash \Box_i\varphi \text{ iff for every } v \in W, R_i wv \text{ implies } \mathcal{M}, v \Vdash \varphi.$$

- It follows that

  $$\mathcal{M}, w \Vdash \Diamond_i\varphi \text{ iff there exists } v \in W \text{ s.t. } R_i wv \text{ and } \mathcal{M}, v \Vdash \varphi.$$

- The definitions of truth in a model ($\mathcal{M} \Vdash \varphi$), of validity in a frame ($\mathcal{F} \Vdash \varphi$) and of the consequence relation are unchanged.

Definition 1.131

A normal multimodal logic is a set 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*

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

  where φ, ψ are formulas and $i \in I$.
- *Λ is closed under generalization: for any formula φ and all $i \in I$,*

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

Multimodal logics

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

Epistemic Logics

Epistemic logics

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

- We consider the multimodal language ML_{Ag}.
- We write, for every $i = 1, \ldots, n$, $K_i\varphi$ 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.

Definition 1.132

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*

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

  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 $\boldsymbol{K}$ the smallest epistemic logic.

Epistemic logics

Recall the following axioms:

$$
\begin{array}{rc}
(T) & K_i\varphi \rightarrow \varphi \\
(D) & \neg K_i(\varphi \wedge \neg\varphi) \\
(B) & \varphi \rightarrow K_i\neg K_i\neg\varphi \\
(4) & K_i\varphi \rightarrow K_iK_i\varphi \\
(5) & \neg K_i\varphi \rightarrow K_i\neg K_i\varphi
\end{array}
$$

Properties of knowledge

- Axiom (T) is called the veridity 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.

Properties of knowledge

- Axiom (B) says that if φ holds, than an agent knows that it does not know $\neg\varphi$.
- 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.

Let $\boldsymbol{S}5 = \boldsymbol{KDB}4 = \boldsymbol{KDB}5 = \boldsymbol{KT}5$. $\boldsymbol{S}5$ is considered as the logic of idealised knowledge.

Theorem 1.133

$\boldsymbol{S}5$ is sound and strongly complete with respect to the class of frames whose relations are equivalence relations.

Reasoning about knowledge

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

Example 1.134

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

We use epistemic logic to make such reasoning precise.

Muddy children puzzle

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

Muddy children puzzle

$k = 1$

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

Muddy children puzzle

$k = 2$

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

Muddy children puzzle

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

Partition model of knowledge

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

Definition 1.135 (Partition frame)

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

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

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

Partition model of knowledge

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

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

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

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

Partition model of knowledge

- For each $i = 1, \ldots, n$, let R_{I_i} be the corresponding equivalence relation.
- Denote by $I_i(w)$ the equivalence class of w in the relation R_{I_i}.
- If the actual world is w, then $I_i(w)$ is the set of possible worlds that agent i cannot distinguish from w.
- $\mathcal{F} = (W, R_{I_i}, \ldots, R_{I_i})$ is a frame for the epistemic logic $\boldsymbol{S}5$.

Partition frame $=$ frame for the epistemic logic $\boldsymbol{S}5$

Partition model of knowledge

Definition 1.136 (Partition model)

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

- $\mathcal{P}_F = (W, I_1, \ldots, I_n)$ *is a partition frame.*
- $\pi : \Sigma \to 2^W$ *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.

Partition model of knowledge

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 1.137 (Logical entailment for partition models)

Let $\mathcal{P}_M = (W, I_1, \ldots, I_n, \pi)$ *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 $\boldsymbol{S}5$

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

Partition model of knowledge

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

- We consider the case $n = k = 2$ (two children, both muddy).
- There are two atomic propositions: muddy1 and muddy2.
- There are four possible worlds, corresponding to each of the children being muddy or not:

$$\begin{aligned}
w_1 &: \quad muddy1 \wedge muddy2 \quad \text{(real world)} \\
w_2 &: \quad muddy1 \wedge \neg muddy2 \\
w_3 &: \quad \neg muddy1 \wedge muddy2 \\
w_4 &: \quad \neg muddy1 \wedge \neg muddy2.
\end{aligned}$$

- Thus, $\pi(muddy1) = \{w_1, w_2\}$ and $\pi(muddy2) = \{w_1, w_3\}$.
- There are two partitions I_1 and I_2.

Partition model of knowledge

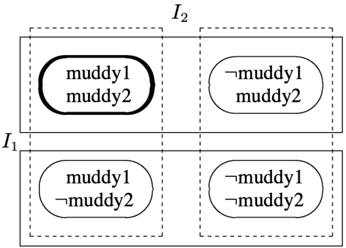


Figure 7: Partition model after the children see each other

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

Partition model of knowledge

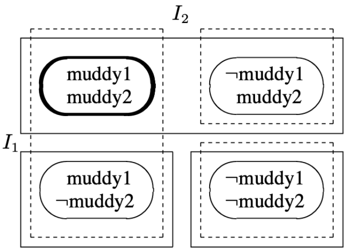


Figure 8: Partition model after the father's announcement

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

Partition model of knowledge

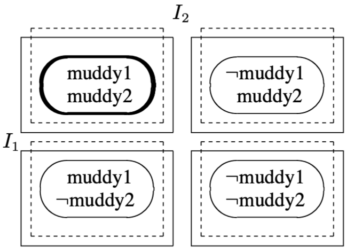


Figure 9: Final partition model

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