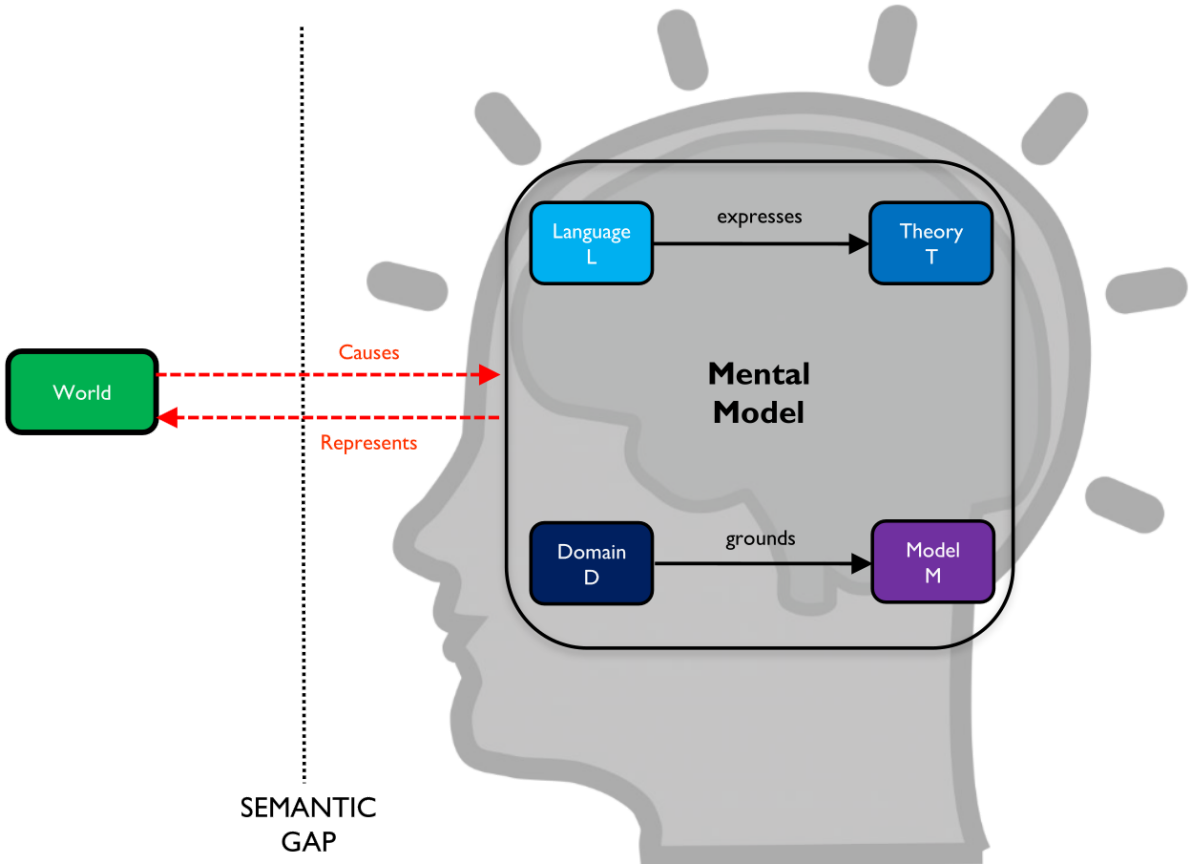
Mathematical logic - Brief

1. Introduction

1.1. Models

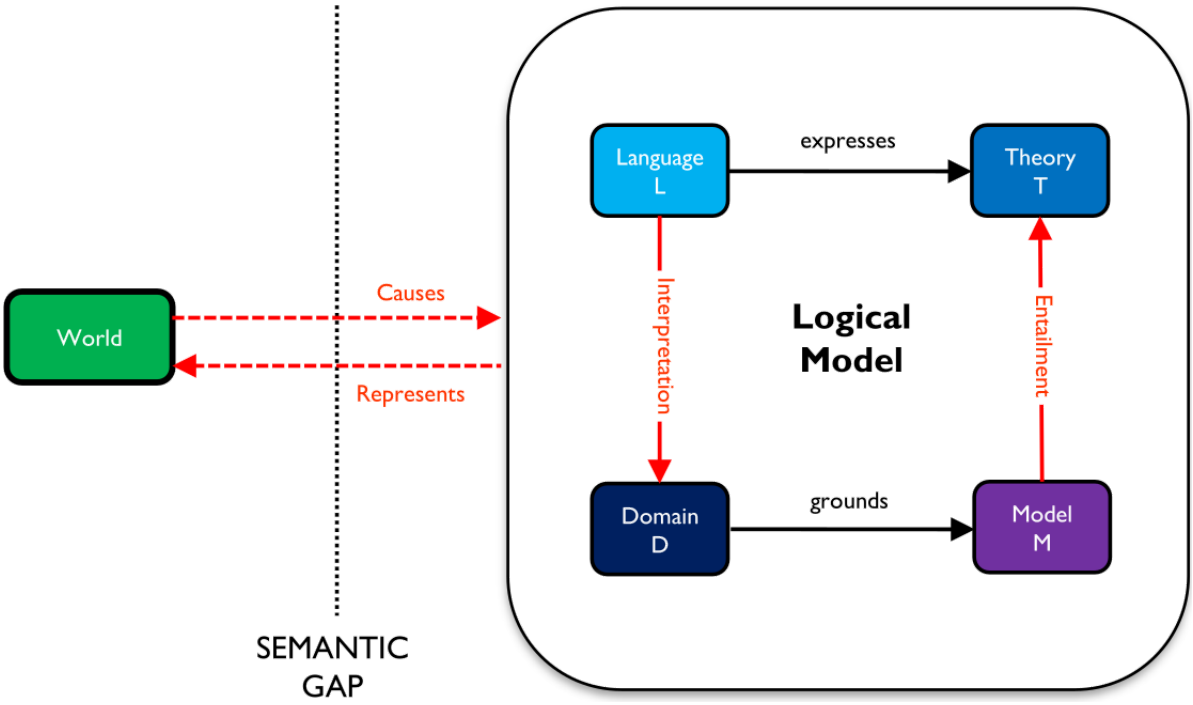
Mental model

> **Mental model**: *human representation of the world*



Logical model

> **Logical model:** *meaning of language made explicit*



- **Interpretation:** $I : L \to D$ (many-to-one)

- **Truth-relation / entailment / satisfiability / ⊨**: relation which associates what is true in the model with a subset of the sentence of the language
  - $S_{(\text{sentence})} \in T \iff \models S$

1.2. Language

Definitions

> **Symbol**: *atomic token*

> **Alphabet**: *set of symbols*

> **Language**: *set of symbols and formation rules to compose them to build correct sentences*

Syntax and semantics

$$\text{language} = \text{syntax} + \text{semantics}$$

> **Syntax**: *the way a language is written*

- Determined by a **set of rules** saying how to construct the expressions of the language from the alphabet

> **Semantics**: *the way a language is interpreted*

- Determines the **meaning of expressions**

> **Expression**: *syntactic construct*

> **Meaning**: *relationship between expressions and elements of some universe of meanings*

1.3. Logical modeling

Formal language

$$\text{formal language} = \text{formal syntax} + \text{formal semantics}$$

> Definition (**Formal syntax**)
>
> - **Alphabet**
> - **Rules for phrase construction**
> - Algorithm for **correctness checking**

> Definition (**Formal semantics**)
>
> - **Interpretation function** $I : L \to D$
>
> Relation between **expressions** in L and **elements** in D
> - **Domain** D
> - **Formal syntax**

Extensional semantics

> **Intended interpretation of L**: *the meanings which are intended to be attached to the symbols and propositions*

> **Extension of a proposition**: *totality, or class, or set of all objects D to which the proposition applies*

Logic

A logic has three **fundamental components**

- **Formal language** (L)
  Defines what can be correctly said
- **Interpretation function** (I)
  Defines how symbols are to be interpreted in intended domain and model
- **Entailment relation** ($\models$)
  Defines/computes two relations in the model
  - **Validity / satisfiability**
  - **Logical consequence**

Theory and model

Given $\{L,\ I,\ \models\}$, a logic allows to define two **components**

> **Theory**: *set of sentences (usually infinite) in L which are assumed true in the intended model, as computed via entailment starting from finite set (called itself theory, or finite presentation of theory)*

> **Model**: *set of facts (usually infinite) expressed in the language describing the mental model (the part of the world observed), in agreement with the theory and the interpretation function*

Decidability and complexity

> **Reasoning**: *deriving consequences of what is known*

Decidability of reasoning

- A **logic is decidable** if there is an **effective method** for determining whether a **formula is included in a theory**
- A **decision procedure** is an algorithm that, given a decision **problem**, terminates with the **correct yes/no answer**
- All logic in this course but first-order logic are decidable

1.4. Formal and informal languages/models

---

2. Set theory

3. Propositional logic

3.1. Intuition

Proposition

> **Proposition**: *statement about what is the case in the world*

- **Atomic language element** of PL
- Can be **true** or **false**
- Different propositions can be **equivalent**

3.2. Language

PL language

Definitions (**Propositional alphabet**)

- **Logical symbols and priority**: $\neg$ > $\wedge$ > $\vee$ > $\rightarrow$ > $\equiv$
- **Non logical symbols**: propositional variables P of set $\texttt{PROP}$
- **Separator symbols**: ( )

Definitions (**Well formed formulas**)

- **Atomic formula**
  - Every $P \in \texttt{PROP}$
- **Formula**
  - Every atomic formula
  - A, B formulas $\Rightarrow$ $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \equiv B$ formulas

Definitions (**Constants and variables**)

- **Constant**: simple proposition
- **Variable**: proposition where a variable can be substituted with a constant, a variable or in general any formula

3.3. Satisfiability

Interpretation of PL

> **Propositional interpretation**: *function that specify if a proposition is true or false*
>
> $$I : \texttt{PROP} \to \{\top, \bot\}$$

- A PI is a **subset** S of $\texttt{PROP}$ such that I is the **characteristic function** of S

$$A \in S \iff I(A) = \top$$

Satisfiability of a formula

> Definition (**Satisfiability**)
>
> A formula A is **satisfied by** (**true in**) an interpretation I (in symbols $I \models A$) according to the following inductive definition
>
> - $I(P) = \top \quad \Rightarrow \quad I \models P, \quad P \in \texttt{PROP}$
>   - $I \not\models A \quad \Rightarrow \quad I \models \neg A$
>   - $I \models A \quad \wedge \quad I \models B \quad \Rightarrow \quad I \models A \wedge B$
>   - $I \models A \quad \vee \quad I \models B \quad \Rightarrow \quad I \models A \vee B$
>   - $I \models A \quad \to \quad I \models B \quad \Rightarrow \quad I \models A \to B$
>   - $I \models A \iff I \models B \quad \Rightarrow \quad I \models A \equiv B$

Checking satisfiability

> Procedure (**Checking satisfiability**)
>
> Given a formula A, of primitive propositions P_i, and an interpretation I

1. **Replace** each occurrence of P_I in A with the truth value assigned by I
2. **Apply** the definitions for connectives

Algorithm (**Lazy evaluation**) (Go to MCDP)

- $A = p$

```
bool check(I ⊨ p)
    if I(p) == true
        return true
    else
        return false
```

- $A = B \wedge C$

```
bool check(I ⊨ B ∧ C)
    if check(I ⊨ B)
        return check(I ⊨ C)
    else
        return false
```

- $A = B \vee C$

```
bool check(I ⊨ B ∨ C)
    if check(I ⊨ B)
        return true
    else
        return check(I ⊨ C)
```

- $A = B \to C$

```
bool check(I ⊨ B → C)
    if check(I ⊨ B)
        return check(I ⊨ C)
    else
        return true
```

- $A = B \equiv C$

```
bool check(I ⊨ B ≡ C )
    if check(I ⊨ B)
        return check(I ⊨ C)
    else
        return ¬(check(I ⊨ C))
```

3.4. Validity and unsatisfiability

Formulas classification (Go to Properties)

Definitions (**Formulas classification**)

A **formula** A is

- **Valid** $\iff \forall I\,, \quad I \models A$
- **Satisfiable** $\iff \exists I \mid I \models A$
- **Unsatisfiable** $\iff \nexists I \mid I \models A$

Propositions (**Implications**)

- $\mathrm{VAL}(A) \Rightarrow \mathrm{SAT}(A) \iff \neg\mathrm{UNSAT}(A)$
- $\mathrm{UNSAT}(A) \iff \neg\mathrm{SAT}(A) \Rightarrow \neg\mathrm{VAL}(A)$

A		$\neg A$
VAL	$\rightarrow$	UNSAT
SAT	$\rightarrow$	$\neg\mathrm{VAL}$
$\neg\mathrm{VAL}$	$\rightarrow$	SAT
UNSAT	$\rightarrow$	VAL

$$\mathrm{VAL}(A) \iff \mathrm{UNSAT}(\neg A)$$
$$\mathrm{SAT}(A) \iff \neg\mathrm{VAL}(\neg A)$$

Theorem (**Refutation principle - validity**) (Please tell me if it is correct)

$$\models A \iff \{\neg A\} \text{ unsatisfiable}$$

Proof

- Suppose that $\models A$
  - $\Rightarrow \quad \forall I,\ I \models A \quad \Rightarrow \quad I \not\models \neg A$
  - $\Rightarrow \quad \nexists I \mid I \models \neg A \quad \Rightarrow \quad \{\neg A\}$ unsatisfiable
- Suppose that $\{\neg A\}$ unsatisfiable
- $\Rightarrow \quad \not\models \neg A \quad \Rightarrow \quad \models A$

Sets of formulas

Definition (**Set of formulas**)

A **set of formulas** Γ is

- **Valid** $\iff \forall A \in \Gamma,\ \forall I,\quad I \models A$
- **Satisfiable** $\iff \forall A \in \Gamma,\ \exists I \mid I \models A$
- **Unsatisfiable** $\iff \forall A \in \Gamma,\ \nexists I \mid I \models A$

Proposition (**Finite set**)

$$\forall_{<\infty} \Gamma = \{A_1, \dots, A_n\},\quad \mathrm{VAL}(\Gamma) \iff \mathrm{VAL}(A_1 \wedge \cdots \wedge A_n)$$
(resp. SAT and UNSAT)

3.5. Logical consequence and equivalence

Definitions

Logical consequence: *a formula A is a LC of a set of formulas Γ iff*

$$\Gamma \models A \iff \forall F \in \Gamma,\ \forall I : I \models F,\quad I \models A$$

Logical equivalence: *two formulas F and G are logically equivalent iff*

$$F \equiv G \iff \forall I,\quad I(F) = I(G)$$

Properties of LC

Γ and Σ are two sets of formulas and A and B two formulas

- **Reflexivity** $\{A\} \models A$
- **Monotonicity** $\Gamma \models A \Rightarrow \Gamma \cup \Sigma \models A$
- **Cut** $\Gamma \models A \wedge \Sigma \cup \{A\} \models B \Rightarrow \Gamma \cup \Sigma \models B$
- **Compactness** $\Gamma \models A \Rightarrow \exists_{<\infty}\, \Gamma_0 \subseteq \Gamma \mid \Gamma_0 \models A$
- **Deduction theorem** $\Gamma, A \models B \iff \Gamma \models A \to B$
- **Refutation principle** $\Gamma \models A \iff \Gamma \cup \{\neg A\}$ unsatisfiable

Theorem (**Refutation principle - LC**)

$$\Gamma \models \varphi \iff \Gamma \cup \{\neg\varphi\} \text{ unsatisfiable}$$

Proof

- Suppose that $\Gamma \models \varphi$
  - $\Rightarrow \quad \forall I \mid I \models \Gamma,\ I \models \varphi \quad \Rightarrow \quad I \not\models \neg\varphi$
  - $\Rightarrow \quad \nexists I \ \mid \ I \models \Gamma \wedge I \models \neg\varphi$
- Suppose that $I \models \Gamma$
  - $\Gamma \cup \{\neg\varphi\}$ unsatisfiable $\Rightarrow \quad I \not\models \neg\varphi \quad \Rightarrow \quad I \models \varphi$

3.6. Axioms and theories

Propositional theory

> **Propositional theory**: *set of formulas closed under the LC relation*
>
> $$T \text{ theory} \iff T \models A \Rightarrow A \in T$$

- A propositional theory always contains an **infinite set of formulas**
  - Any theory T contains at least **all the valid formulas**, which are infinite

> **Logical closure**: *for any set of formulas* Γ
>
> $$\mathrm{cl}(\Gamma) = \{A \mid \Gamma \models A\}$$

- For any set Γ, the closure $\mathrm{cl}(\Gamma)$ is a **theory**

Axiomatization

> **Set of axioms**: *a set of formulas* Ω *is a set of axioms for a theory* T *if*
>
> $$\forall A \in T, \ \ \Omega \models A$$

- Γ is a set of axioms for $\mathrm{cl}(\Gamma)$

> **Finitely axiomatizable theory**: *a theory* T *is finitely axiomatizable if it has a finite set of axioms*

Logic based systems

- Used for **representing** and **reasoning** about **knowledge**
- Composed by
  - **Reasoning system**
  - **Knowledge base**
    - Consists of a finite **collection of formulas** in a logical language
    - **Answers queries** submitted to it by means of a **reasoning system**

- **Tell** action
  - Incorporates the **new knowledge** encoded in an **axiom** (formula)
  - Allows to build a **knowledge base**
- **Ask** action
  - Allows to **query what is known**
    - If a formula φ is a LC of the axioms contained in the KB ($KB \models \varphi$)

4. Reasoning via truth tables

4.1. Summary

4.2. Decision problems

Reasoning / Decision problems

Four types of **questions**

- **Model checking** $\quad \mathrm{MC}(I, \varphi) \ : \ I \models \varphi$
  What is the truth value of φ in I ?
- (Un)**Satisfiability** $\quad \mathrm{SAT}(\varphi) \ : \ \exists I \mid I \models \varphi$
  Is there a model I that satisfies φ ?
- **Validity** $\quad \mathrm{VAL}(\varphi) \ : \ \forall I, \ I \models \varphi$
  Is φ satisfied by all the models I ?
- **Logical consequence** $\quad \mathrm{LC}(\Gamma, \varphi) \ : \ \Gamma \models \varphi$
  Is φ satisfied by all the models I that satisfy all the formulas in Γ ?

Model checking

> Definition (**MC decision procedure**)
>
> Algorithm that checks if a **formula** φ is satisfied by an **interpretation** I
>
> $$\begin{aligned} \mathrm{MCDP}(\varphi, I) &= \top \iff I \models \varphi \\ \mathrm{MCDP}(\varphi, I) &= \bot \iff I \not\models \varphi \end{aligned}$$

Algorithm (**MCDP naive**)

1. **Replace** each occurrence of a variable P in φ with the truth value $I(P)$
2. Recursively apply the **reduction rules** for connectives

Algorithm (**MCDP lazy evaluation**)

Use lazy evaluation rules to skip **superfluous evaluations**

- $A = p$

```
bool MCDP(I, p)
    if I(p) == true
        return true
    else
        return false
```

- … see satisfiability rules

Satisfiability

Definition (**Satisfiability decision procedure**)

Algorithm that takes in input a **formula** φ and checks if φ is (un)satisfiable

$$\begin{aligned} \mathrm{SDP}(\varphi) &= \text{satisfiable} &&\iff \exists I \mid I \models \varphi \\ \mathrm{SDP}(\varphi) &= \text{unsatisfiable} &&\iff \nexists I \mid I \models A \end{aligned}$$

- If $\mathrm{SDP}(\varphi) =$ satisfiable, it can **return at least a model** I that satisfies φ

Validity

> Definition (**Validity decision procedure**)
>
> Algorithm that checks if a **formula** is valid
>
> VDP can be based on SDP by exploiting the **refutation principle (VAL)**
>
> $$\begin{array}{lllll} \mathrm{VDP}(\varphi) & = & \top & \iff & \mathrm{SDP}(\neg\varphi) & = & \text{unsatisfiable} \\ \mathrm{VDP}(\varphi) & = & \bot & \iff & \mathrm{SDP}(\neg\varphi) & = & \text{satisfiable} \end{array}$$

- Interpretation I returned by $\mathrm{SDP}(\neg\varphi)$ is a **counter-model** for φ

Logical consequence

> Definition (**LC procedure**)
>
> Algorithm that checks whether a **formula** φ is a LC of a **finite set of formulas** $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$
>
> LDCP can be based on SDP by exploiting the **refutation principle (LC)**
>
> $$\begin{array}{lllll} \mathrm{LCDP}(\Gamma, \varphi) & = & \top & \iff & \mathrm{SDP}(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\varphi) & = & \text{unsatisfiable} \\ \mathrm{LCDP}(\Gamma, \varphi) & = & \bot & \iff & \mathrm{SDP}(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\varphi) & = & \text{satisfiable} \end{array}$$

- Interpretation I returned by $\mathrm{SDP}(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\varphi)$ is a **model** for Γ and a **counter-model** for φ

4.3. Conjunctive normal form

Definitions

> **Literal**: *either a variable or the negation of a variable*

> **Clause**: *disjunction of literals.*

> **Conjunctive normal form**: *conjunction of clauses*
>
> $$(I_{11} \vee \cdots \vee I_{1n_1}) \wedge \cdots \wedge (I_{m1} \vee \cdots \vee I_{mn_m}) \equiv \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{n_j} I_{ij} \right)$$

Properties

Properties of $\wedge$ and $\vee$

- **Commutativity** $\varphi * \psi \equiv \psi * \varphi$
- **Absorption** $\varphi * \varphi \equiv \varphi$

Properties of clauses

- **Order** of literals does not matter
- Multiple literals can be **merged**
- Clauses as **set of literals**

Properties of formulas in CNF

Same as **clauses** (with the right substitution of terms)

In addition

- **Existence**: every formula can be reduced into CNF

- **Equivalence:** $\models \mathrm{CNF}(\varphi) \equiv \varphi$

Reduction in CNF

> **CNF function:** *recursive function that transforms a formula in its CNF*
>
> See slide 9

Examples slides 10-12

4.4. DPLL SAT decision procedure

Satisfiability of a set of clauses

> **Partial evaluation:** *partial function that associates to some variables of the alphabet* `PROP` *a truth value and can be undefined for the other elements*

- Allows to **construct models** for a set of clauses **incrementally**
- To check satisfiability, we don't need to know all the truth values

Partial valuation

> Procedure (**DPLL**)
>
> 1. Starts with an **empty valuation** (undefined for all letters)
> 2. **Extends** it step by step to letters in N

DPLL

Procedure (**Simplification of a formula** by an evaluated literal)

For any CNF formula φ and atom p, the notation $\varphi|_p$ stands for the formula obtained from φ by

1. **Replacing** all occurrences of p by $\top$
2. **Simplifying** the result by removing
    - **Clauses** containing the disjunctive term $\top$
    - **Literals** $\neg\top = \bot$ in all remaining clauses

Similarly $\varphi|_{\neg p}$ is the result of

1. Replacing p in φ by $\bot$
2. Simplifying the result, according to the process dual to above

Unit clause: *CNF formula φ that contains a single-literal clause $C = \{l\}$*

Procedure (**Unit propagation**)

Simplification of φ

```
void UnitPropagation(φ, I)  // I(φ) initially empty
    while φ ⊇ {l}
        φ = φ|l
    if l == p
        I(p) = true
    if l == ¬p
        I(p) = false
```

- **Outputs**

    - $\{\}$ $\Rightarrow$ satisfiable
    - $\{\{\}, \cdots\}$ $\Rightarrow$ unsatisfiable
- **UP does not terminate** when the CNF
    - Isn't in the **previous forms**
    - Does not contain **unit clauses**

DPLL definition

Procedure (**DPLL**)

Extension of the UP method that can solve the satisfiability

```
bool DPLL(φ, I)  // I(φ) initially empty
    UnitPropagation(φ, I)
    if φ ⊃ {}
        return false
    if φ = {}
        return true  // exit with I

    // heuristic choice of l in order to achieve
efficiency
    l = selectLiteralIn(C ∈ φ)

    // l == p  → ln = ¬p
    // l == ¬p → ln = p
    return (DPLL(φ|l , I ∪ (I(l) = true)) ||
            DPLL(φ|ln, I ∪ (I(l) = false)))
```

5. PL - reasoning as deduction

5.1. Reasoning as deduction

Deduction / Proof

> Definition (**Deduction / Proof**)
>
> Given
>
> 1. **Premises** Γ
> 2. **Conclusion** A
>
> A deduction is a **sequence** or a **tree/Direct Acyclic Graph** of nodes, where
>
> - Each **node** of the deduction is labeled with a **formula**
> - **Links** are labeled with **motivation** (inference rules)
> - **Root** nodes are **premises**
> - **Leaf** nodes are **conclusion**
>
> $\Gamma \vdash A$ means that (equivalently)
>
> - There is at least a **deduction** which connects Γ and A
> - A can be **deduced**/**derived** from Γ

Key properties that have to be satisfied

- **Correctness theorem** ($\Rightarrow$)
- **Completeness theorem** ($\Leftarrow$)

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

5.2. Hilbert systems (VAL - forward chain)

5.3. Tableaux systems (SAT - backward)

Tableaux calculus and method

> **Tableaux calculus**: *algorithm solving the problem of satisfiability*

- Tableaux **rooted in** φ is a method to search **interpretations that satisfies** φ
- Tableaux exaustively builds a branch for **any possible truth assignment**
  - **Interpretation** corresponding to a **branch** should **satisfy all the formulas** that appear in the branch

Tableau method allows to perform **automated deduction**

- Given: set of premises Γ and conclusion φ
- Task: prove $\Gamma \models \varphi$
- How: show $\Gamma \cup \{\neg\varphi\}$ is not satisfiable via **refutation procedure**

> **Refutation procedure**: *add the complement of the conclusion to the premises and derive a contradiction*

Constructing proofs

- **Data structure**: a proof/deduction is represented as a tableau
  - Binary tree which nodes are labeled with formulas
- **Start**: put premises and negated conclusion into root of empty tableau
- **Expansion**: apply expansion rules to the formulas on the tree
  - Add new formulas and split branches
- **Closure**: close branches that are obviously contradictory
- **Success**: a proof is successful iff we can close all branches

Expansion rules (Go to FO tableaux)

Definition (**Standard / Smullyan-style tableau rules**)

α rules

$$\frac{\varphi \wedge \psi}{\begin{array}{c}\varphi\\ \psi\end{array}} \qquad \frac{\neg(\varphi \vee \psi)}{\begin{array}{c}\neg\varphi\\ \neg\psi\end{array}} \qquad \frac{\neg(\varphi \to \psi)}{\begin{array}{c}\varphi\\ \neg\psi\end{array}}$$

$\neg\neg$ elimination

$$\frac{\neg\neg\varphi}{\varphi}$$

β rules

$$\frac{\varphi \vee \psi}{\varphi \mid \psi} \qquad \frac{\neg(\varphi \wedge \psi)}{\neg\varphi \mid \neg\psi} \qquad \frac{\varphi \to \psi}{\neg\varphi \mid \psi}$$

Branch closure

$$\frac{\begin{array}{c}\varphi\\ \neg\varphi\end{array}}{\mathsf{X}}$$

$$\varphi \left\langle \equiv, \iff \right\rangle \psi \quad := \quad (\varphi \to \psi) \wedge (\psi \to \varphi)$$

Smullyan's uniform notation

Two **types of formulas** / rules

- **Conjunctive** / determininstic (α)
- **Disjunctive** / splitting (β)

α	α_1	α_2
$\varphi \wedge \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \to \psi)$	φ	$\neg\psi$

β	β_1	β_2
$\varphi \vee \psi$	φ	ψ
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\varphi \to \psi$	$\neg\varphi$	ψ

α and β rules can be **stated** as follows

$$\frac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\beta_1 \mid \beta_2}$$

Definitions

> **Closed branch**: *branch which contains a formula and its negation* (X)

> **Open branch**: *branch which isn't closed* (O)

> **Closed tableaux**: *tableaux with all branches closed*

> **Derivation / LC**: *given a formula φ and a finite set of formulas Γ, $\Gamma \vdash \varphi$ means that there exists a closed tableau for $\Gamma \cup \{\neg\varphi\}$*

Test validity and satisfiability

A tableaux for Γ builds an **interpretation** for Γ

Construct tableaux to test

- **Satisfiability of a set Γ**
  - Root: **all formulas** in Γ
    - Close off $\rightarrow$ Γ **not satisfiable**
    - Not close off $\rightarrow$ Γ **satisfiable**
- **Validity of a formula φ**
  - Root: $\neg\varphi$
    - Close off $\rightarrow$ φ logically **valid**
- φ is a **logical consequence** of Γ
  - Root: **all formulas** in Γ and $\neg\varphi$
    - Close off $\rightarrow$ φ **LC** of Γ
- **Logical equivalence of two formulas**
  - If **LC** holds in **both direction**

φ	$\neg\varphi$	φ	$\neg\varphi$	φ	$\neg\varphi$
X X	X X	O X	O X	O O	O O
$\neg\mathrm{SAT}$	VAL	SAT	$\neg\mathrm{VAL}$		
$\downarrow$	$\downarrow$	$\wedge$	$\wedge$	SAT	$\neg\mathrm{VAL}$
$\neg\mathrm{VAL}$	SAT	$\neg\mathrm{VAL}$	SAT		

Build interpretations

For each **open branch** and for each **atom** p in the formula, $I(p)$ is defined as

$$I(p) = \begin{cases} \top & \text{if } p \text{ belongs to the branch} \\ \bot & \text{if } \neg p \text{ belongs to the branch} \end{cases}$$

If neither p nor $\neg p$ belong to the branch, $I(p)$ is defined in **arbitrary way**

Soundness and completeness

> Theorem (**Soundness**)
>
> $$\Gamma \vdash \varphi \;\Rightarrow\; \Gamma \models \varphi$$

> Theorem (**Completeness**)
>
> $$\Gamma \models \varphi \;\Rightarrow\; \Gamma \vdash \varphi$$

Decidability

> Theorem (**Decidability**)
>
> Tableau method is a **decision procedure** for classical PL
>
> Proof
>
> To **check validity** of φ, develop a tableau for $\neg\varphi$
>
> Because of **termination**, we will eventually get a tableau that is either closed (1) or that has a branch that cannot be closed (2)
>
> 1. The formula φ must be valid (**soundness**)
> 2. The branch that cannot be closed shows that $\neg\varphi$ is satisfiable, so φ cannot be valid (**completeness**)

7. First order logic

7.1. Intuition

First order logic

> Definitions (**FOL constructs**)
>
> **FOL** assumes world contains
>
> - **Constants**
> - **Predicates**
> - **Functions**
>
> In FOL it is possible to **build**
>
> - **Atomic propositions**
>   - Applying **predicate to constants**
>
> $$\text{Predicate}(\text{Constants})$$
>
> - **Propositions**
>   - Applying **function to a constant**
>   Then **predicate to the resulting** object
>
> $$\text{Predicate}(\text{Function}(\text{Constant}))$$
>
>   - Applying universal (existential) **quantifiers to variables**
>   This allows to **quantify to arbitrary objects** of the universe
>
> $$[\text{Qt}]\ x.\ \text{Proposition}(x)$$

7.2. Language

Syntax

Definition (**Alphabet**)

- **Logical symbols**
  - **Constant** $\bot$
  - **PL connectives** $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\equiv$
  - **Quantifiers** $\forall$, $\exists$
  - Infinite set of **variable symbols** $x_1, x_2, \ldots$
  - **Equality symbol** $=$ (optional)
- **Non logical symbols**
  - a set $c_1, c_2, \ldots$ of **constants symbols** [1]
  - a set $f_1, f_2, \ldots$ of **functional symbols** [2]
  - a set $P_1, P_2, \ldots$ of **relational symbols** [2]

- **Non logical symbols**
  - Depends from the **domain** we want to model
  - Must have an **intuitive interpretation** on such a domain

Terms and formulas

Definition (**Language**)

- **Terms**

    - Every **constant** c_i and every **variable** x_i
    - $t_1, \ldots, t_n$ terms, f_i functional $| \#f_i = n \Rightarrow f(t_1, \ldots, t_n)$ term
- (Well formed) **Formulas**
    - t_1, t_2 terms $\Rightarrow t_1 = t_2$ formula
    - $t_1, \ldots, t_n$ terms, P_i relational $| \#P_i = n \Rightarrow P_i(t_1, \ldots, t_n)$ formula
    - A, B formulas $\Rightarrow \langle \bot, A \wedge B, A \to B, A \vee B, \neg A, A \equiv B \rangle$ formulas
    - A formula, x variable $\Rightarrow \langle \forall x.\, A, \exists x.\, A \rangle$ formulas

Common mistakes

- **Use of $\wedge$ with $\forall$**
- **Use of $\to$ with $\exists$**

Representing variations quantifiers

Definition (**At least n**)

$$\exists x_1 \cdots \exists x_n \left(\bigwedge_{i=1}^{n} \varphi(x_i) \wedge \bigwedge_{i \neq j = 1}^{n} x_i \neq x_j \right)$$

Definition (**At most n**)

$$\forall x_1 \cdots \forall x_{n+1} \left(\bigwedge_{i=1}^{n+1} \varphi(x_i) \to \bigvee_{i \neq j = 1}^{n+1} x_i = x_j \right)$$

7.3. Interpretation function

Interpretation function

Definition (**Interpretation for a language** L)

A FO interpretation for the language

$$L = (c1,\, c2,\, \ldots,\, f1,\, f2,\, \ldots,\, P1,\, P2,\, \ldots)$$

is a **pair** $(\Delta,\, I)$ where

- Δ is a non empty set called **interpretation domain**
- I is is a function, called **interpretation function**
  - $I(c_i) \in \Delta$ (elements of the domain)
  - $I(f_i) : \Delta^n \to \Delta$ (n-ary function on the domain)
  - $I(P_i) \subseteq \Delta^n$ (n-ary relation on the domain)

  Where n is the **arity** of f_i and P_i

Example slides 4-5

7.4. Satisfiability w.r.t an assignment

Interpretation of terms

Definition (**Assignment**)

Function a from set of variables to domain of interpretation Δ

$$a[x/d]$$

Denotes the **assignment that coincides** with a on all the variables but x, which is associated to d

Definition (**Interpretation of terms**)

The **interpretation** I of a term t w.r.t. the assignment a, in symbols $I(t)[a]$, is **recursively defined** as follows

$$\begin{aligned} I(x_i)[a] &= a(x_i) \\ I(c_i)[a] &= I(c_i) \\ I\big(f(t_1,\ldots,t_n)\big)[a] &= I(f)\big(I(t_1)[a],\ \ldots,\ I(t_n)[a]\big) \end{aligned}$$

Example slide 3

Satisfiability of formulas

Definition (**Satisfiability of a formula w.r.t.** a)

An interpretation I satisfies a formula φ **w.r.t. the assignment** a according to the following **rules**

$$\begin{array}{lll} I \models t_1 = t_2 & [a] & \iff I(t_1)[a] = I(t_2)[a] \\ I \models P(t_1,\ldots,t_n) & [a] & \iff \langle I(t_1)[a],\ldots,I(t_n)[a]\rangle \in I(P) \\ I \models \varphi \wedge \psi & [a] & \iff I \models \varphi[a] \wedge I \models \psi[a] \\ I \models \varphi \vee \psi & [a] & \iff I \models \varphi[a] \vee I \models \psi[a] \\ I \models \varphi \to \psi & [a] & \iff I \not\models \varphi[a] \vee I \models \psi[a] \\ I \models \neg\varphi & [a] & \iff I \not\models \varphi[a] \\ I \models \varphi \equiv \psi & [a] & \iff I \models \varphi[a] \iff I \models \psi[a] \\ I \models \exists x.\,\varphi & [a] & \iff \exists\, d \in \Delta \mid I \models \varphi[a[x/d]] \\ I \models \forall x.\,\varphi & [a] & \iff \forall d \in \Delta,\ I \models \varphi[a[x/d]] \end{array}$$

Example slides 6-7

7.5. SAT, VAL, LC, LE

Free variables and terms

> Definition (**Free occurrence**)
>
> A free occurrence of a variable x is an occurrence of x which is **not bounded by** a (universal or existential) **quantifier**
>
> - Any occurrence of x in t_i is free in $P(t_1, \ldots, t_n)$
> - Any f.o. of x in φ or in ψ is also free in $\varphi \wedge \psi$, $\psi \vee \varphi$, $\psi \to \varphi$, $\neg\varphi$
> - Any f.o. of x in φ is free in $\forall y.\, \varphi$ and $\exists y.\, \varphi$ if y is distinct from x

> **Ground formula**: *formula that doesn't contain any variable*

> **Closed formula**: *formula that doesn't contain f.o. of variables*

> **Free variable**: *variable x is free in φ (denote by $\varphi(x)$) if there is at least a free occurrence of x in φ*

- Free variables represent **individuals** which must be **instantiated** to make the formula a meaningful proposition

> **Free term**: *term t is free for variable x in formula φ iff all the occurrences of x in φ don't occur in the scope of a quantifier of some variables in t*

Satisfiability and validity

> Definition (**Model, satisfiability and validity**)
>
> - An interpretation I is a **model** of φ under the assignment a if

$$I \models \varphi[a]$$

- A formula φ is **satisfiable** if

$$\exists\, I,\, \exists\, a \;\mid\; I \models \varphi[a]$$

- A formula φ is **unsatisfiable** if it is not satisfiable

$$\neg\big(\exists\, I,\, \exists\, a \;\mid\; I \models \varphi[a]\big)$$

- A formula φ is **valid** if

$$\forall\, I,\, \forall\, a\,,\;\; I \models \varphi[a]$$

Definition (**Logical consequence**)

A formula φ is a LC of a set of formulas Γ, in symbols $\Gamma \models \varphi$, if

$$\forall\, I,\, \forall\, a\,,\;\; I \models \Gamma[a] \;\Rightarrow\; I \models \varphi[a]$$

- $I \models \Gamma[a]$ means that I satisfies all the formulas in Γ under a

Logical equivalence defined as **bidirectional LC**

Open and closed formulas

For **closed formulas**

- SAT, VAL, LC, LE **don't depend on the assignment** of variables
- We can **omit the assignment** and write $I \models \varphi$

In general

$$I \models \varphi[a] \iff I \models \varphi[a']$$

when $[a]$ and $[a']$ coincide on the variables free in φ

- $[a]$ and $[a']$ can **differ** on all the other variables

- With **closed formulas** holds for **all assignments**

7.6. Exercises

7.7. Finite domains

FD with names for every element

> **Unique name assumption**: *assumption under which the language contains a name for each element of the domain*
>
> $$\text{UNA} := \varphi_{\Delta=\{c_1,\dots,c_n\}} = \left(\bigwedge_{i\neq j=1}^{n} c_i \neq c_j \wedge \forall x \left(\bigvee_{i=1}^{n} x = c_i \right) \right)$$

- The language contains the constants $c_1, \dots, c_n$, and each constant is the name of **one and only** one domain element
- Finite domain with a **constant name** for every elements
- Constants are also elements of the **domains**

Finite predicate extension

The **assumption** that states that a **predicate P is true** only for a finite set of objects for which the language contains a name, can be formalized as

$$\forall x \big(P(x) \equiv (x = c_1 \vee \cdots \vee x = c_n) \big)$$

Example slide 4

Grounding

Under the hypothesis of FD with the UNA, **FO formulas** can be **propositionalized** (**grounded**) as

$$\varphi_{\Delta=\{c_1,\ldots,c_n\}} \models \forall x \big(\varphi(x) \equiv \varphi(c_1) \wedge \cdots \wedge \varphi(c_n) \big)$$

$$\varphi_{\Delta=\{c_1,\ldots,c_n\}} \models \exists x \big(\varphi(x) \equiv \varphi(c_1) \vee \cdots \vee \varphi(c_n) \big)$$

Generalizing

$$\varphi_{\Delta=\{c_1,\ldots,c_n\}} \models \forall x_1 \cdots \forall x_k \left(\varphi(x_1, \ldots, x_k) \equiv \bigwedge_{\substack{c_{i_1},..,c_{i_k} \in \\ \in \{c_1,..,c_n\}}} \varphi(c_{i_1}, \ldots, c_{i_k}) \right)$$

$$\varphi_{\Delta=\{c_1,\ldots,c_n\}} \models \exists x_1 \cdots \exists x_k \left(\varphi(x_1, \ldots, x_k) \equiv \bigvee_{\substack{c_{i_1},..,c_{i_k} \in \\ \in \{c_1,..,c_n\}}} \varphi(c_{i_1}, \ldots, c_{i_k}) \right)$$

Grounding allows to **reduce FOL reasoning to PL reasoning**

7.8. Analogies with databases

Query answering in FOL

- Purpose: knowing the **set of objects** which share a **given property**
- General purpose: knowing the set of n**-tuples of objects** which are in a certain n**-ary relation**

- A **property in FOL** can be expressed by a **formula** $\varphi(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$

> Definition (**Query answering**)
>
> Given
>
> - **Interpretation** I (a database instance) of a FOL language
> - **Formula** $\varphi(x_1, \ldots, x_n)$ with n free variables
>
> Find all the n**-tuples** of elements of domain $(d_1, \ldots, d_n) \in (\Delta^I)^n$ such that
>
> $$I \models \varphi\big[a[x_1/d_1 \cdots x_n/d_n]\big]$$

Analogy with relational DBs

1. $\varphi(x_1, \ldots, x_n)$ represents one DB **relation**
2. Relational **DB** as a **set of formulas**

Examples slides 3, 5-6

Requirements

FOL can be used to **formalize relational databases** iff

- **Domain** of interpretation Δ is **finite**
- There is the **UNA**
- L contains **no functional symbols** (relational language)
- **Unknown facts** (not stated in tables) are assumed to be **false** (Closed World Assumption)

Correspondences

FOL	Relational DB
Non logical symbols of L	Database schema [3]
Domain Δ [4]	Set of values which appears in the tables
Interpretation I of a relation	Tuples that belong to each relation
“Certain” formulas on L	Queries over the database
Interpretation of formulas of L [5]	Answers

8. FOL - reasoning as deduction

8.1. Reasoning problems (recap)

8.2. Hilbert systems (VAL - forward chain)

8.3. Tableaux systems (SAT - backward)

First order tableaux

Definition (**First order tableaux**)

A tableau is a **rooted tree**, where each **node** carries a **FO sentence** (closed formula), and the **children** of a node n are generated by applying a set of **expansion rules** to n or to one of the ancestors of n

Definition (**Expansion rules**)

The expansion rules for a **FO semantic tableaux** are

- α and β **rules** (See propositional semantic tableaux)

α rules

$$\frac{\varphi \wedge \psi}{\begin{array}{c}\varphi\\ \psi\end{array}} \qquad \frac{\neg(\varphi \vee \psi)}{\begin{array}{c}\neg\varphi\\ \neg\psi\end{array}} \qquad \frac{\neg(\varphi \to \psi)}{\begin{array}{c}\varphi\\ \neg\psi\end{array}}$$

β rules

$$\frac{\varphi \vee \psi}{\varphi \mid \psi} \qquad \frac{\neg(\varphi \wedge \psi)}{\neg\varphi \mid \neg\psi} \qquad \frac{\varphi \to \psi}{\neg\varphi \mid \psi}$$

- Extended with rules that deal with the **quantifiers**

γ rules

$$\frac{\forall x.\, \varphi(x)}{\varphi(t)} \qquad \frac{\neg\exists x.\, \varphi(x)}{\neg\varphi(t)}$$

δ rules

$$\frac{\neg\forall x.\, \varphi(x)}{\neg\varphi(c)} \qquad \frac{\exists x.\, \varphi(x)}{\varphi(c)}$$

  - t is a **term free** for x in φ
  - c is a **new fresh constant** not previously appearing in the tableaux

Fresh constant/variable: *term that denote an unconditioned object for denoting an unknown object*

Substitution

> **Substitution**: $\varphi[x/t]$ *denotes the formula we get by replacing each free occurrence of the variable* x *in the formula* φ *by the term* t

- This is admitted if t does **not contain any variable** y such that x occurs in the scope of a quantifier for y
- If $\varphi(x)$ is a **free variable** and t is a **term**, we use the notation $\varphi(t)$ (instead of the more precise $\varphi[x/t]$) to represent the **substitution** of x for t in φ

8.4. Examples

Examples PDF 82 slides 13-16

Examples PDF 84

8.5. Termination

Termination of a FOL tableaux

In contrast to PL, FOL tableaux construction is **not guaranteed to terminate**

If the **formula** φ that labels the **root** is

- **Unsatisfiable**, then construction

    - Is **guaranteed to terminate** and tableau can be **closed**
- **Satisfiable**, then construction either
    - Is **guaranteed to terminate** and tableau is **open**
    - Does **not terminate**

> Problem (**Search dilemma**)
>
> If you have **not** yet been able to **close** the tableaux, is it either
>
> - because the formula is **satisfiable**?
> - because you have **not found the way** to construct the tableaux?
>
> You cannot know!

8.6. Counter models

Saturated branches

> γ-**formulas**: *formulas in the form* $\forall x \varphi$ *or* $\neg \exists x \varphi$

> Definition (**Saturated open branch**)
>
> **Open branch** where
>
> - Every **non-literal** has been **analyzed** at least once
> - Every γ-**formula** has been **instantiated with every term** we can construct using the function symbols on the branch

Failing proof: tableau with an SOB can never be closed

- We can **stop** an declare the proof a failure

Counter models

- If the construction of a tableaux ends in a **saturated open branch**, the
  tableaux can be used to **define the interpretation** which is also a model
  M for all the formulas on that branch
- M is **finite by construction**, it is a subset of other possibly infinite models
- A **model** M, being an interpretation, must tell how to **interpret constants** (the elements of the domain), function symbols, and predicate symbols
- **Domain**: set of all terms constructible using function symbols on branch
- **Function symbols**: interpreted as themselves (or using the fake constants)
- **Predicate symbols**: interpreted in terms of their occurrences in the branch

---

1. Functions with **arity equal to 0** ↩

2. Each of which is associated with its **arity** (number of arguments) ↩ ↩

3. (tables) ↩

4. Elements of the language and of the domain have the same name ↩

5. (queries) ↩