Mathematical logic - Full

1. Introduction

1.1. Models

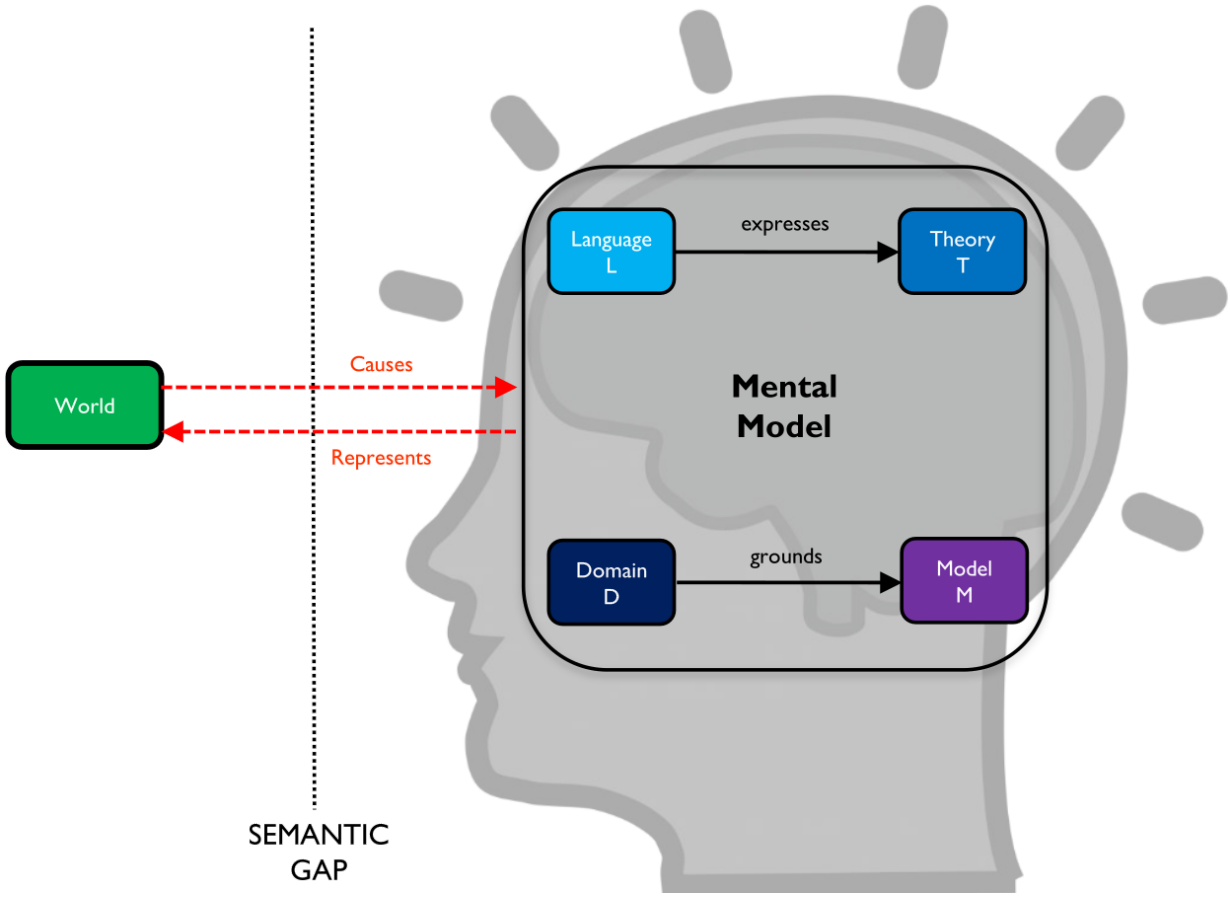
Conceptual models

> **Conceptual model**: *meaningful representation of a portion of the world, described in a certain language*

> **Conceptual modeling**: *activity which leads to the construction of models*

Mental model

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

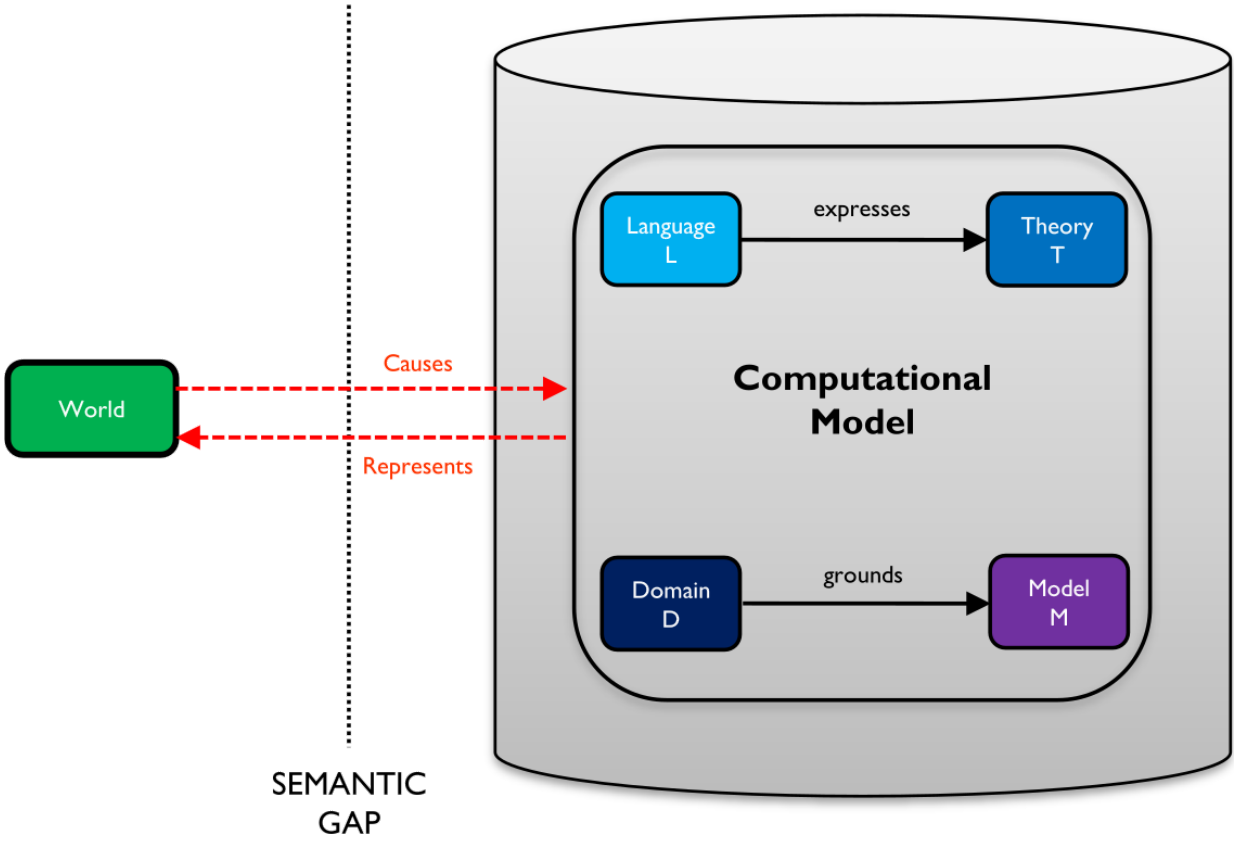


- **World**: what we perceive
- **Mental model**: mental representation of world, decomposed in 4 constituents
  - **Language**: alphabet + syntax used to describe the world
  - **Theory** (**fact**): sentences describing what is true in the world
  - **Domain**: images which represent atomic elements used to describe what we see
  - **Model**: images which represent the sets of facts (scenes) that we see
- **Semantic gap**: difference between world and mental model

The **link** from language to the images is **in the mind of the person** looking at the world, but not in the mind of the others

Computational model

> **Computational model**: *implementation of programs*

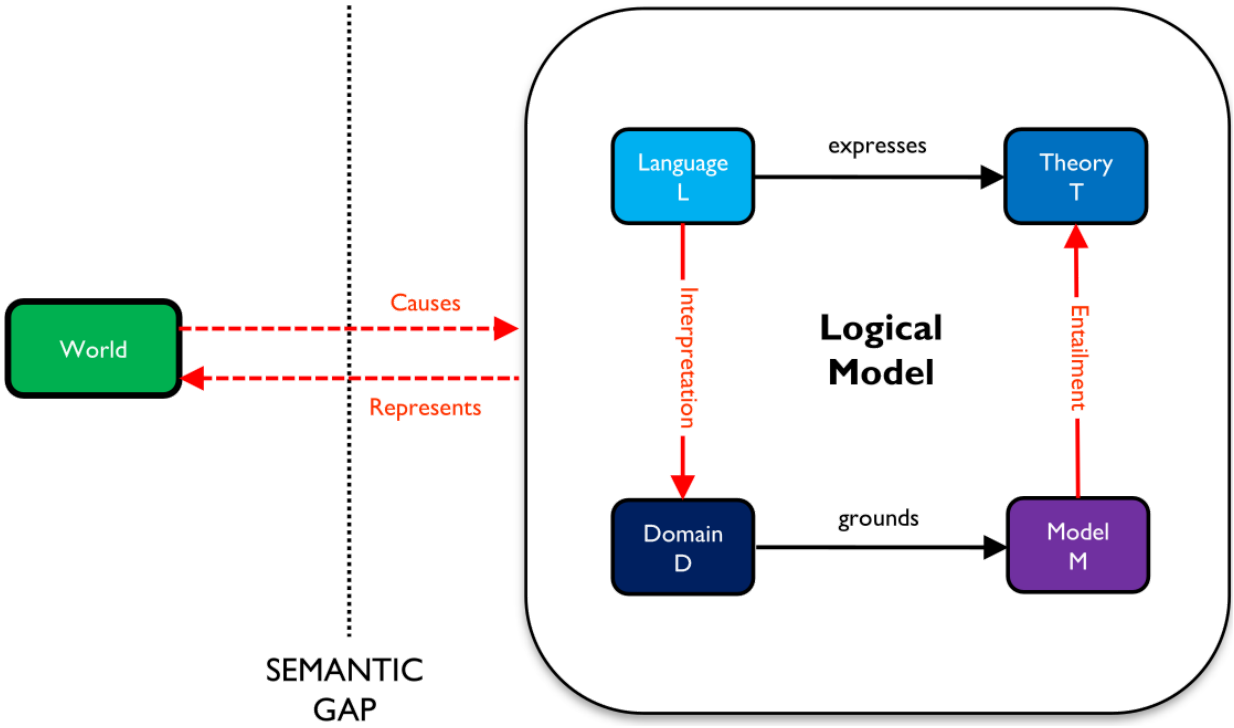


- **Language**: abstraction of the language of the table
- **Theory**: actual table
- **Domain**: possible memory locations
- **Model**: actually used memory locations

No computer mental model: **mental model** is only in the mind of **developer**

Logical model

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



- **Interpretation**: function which associates each element of the language to one and only one element of the domain
- **Truth-relation** / ***entailment*** [1] **/ satisfiability** / ⊨: relation which associates what is true in the model with a subset of the sentence of the language
  - A **sentence** can be an element in a **theory** if and only if its interpretation is true in the model

Logical model can be used to **describe** both **mental** and **computational** ones

Mental	Computational	Logical
Images	Memory locations	Sets of elements
Always finite	Always finite	Often infinite

Usefulness

Logical models make **precise** what we mean when we describe something

- **Interaction** developer-customer
  - Implementation of requirements, change safe
- **Integration** between two **developers/programs**
  - Input/output *compliance* [2]
    - Syntactic compliance (standards)

        - Semantic/meaning compliance
- **High value applications** because of its cost
    - Largely solved, lots of solutions in the market
- **System interoperability** (web)

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*

- **Sentences** are usually unbound in length, but still **finite**

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*

- **Universe** may or may not be **formalized**

1.3. Logical modeling

Formal language

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

> Definition (**Formal syntax**)
>
> - Infinite/finite (always recognizable) **alphabet**
> - Finite set of formal constructors and building **rules for phrase construction**
> - Algorithm for **correctness checking** (a phrase in a language)

> Definition (**Formal semantics**)
>
> - **Interpretation function** (relationship) I between syntactic **constructs** in a language L and the **elements** of an universe of meanings D
>
> $$I : L \to D$$
>
> - **Domain** D is always as a **set of elements**
> - Requires **formal syntax**

> Definition (**Informal syntax/semantics**)
>
> The **opposite** of formal (absence of the elements above)

Examples slides 4-6

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 the intended domain and model
- **Entailment relation** ($\models$): defines/computes two relations (in the model)
  - **Validity/satisfiability**: truth of a formula
  - **Logical consequence**: the truth of a set of formulas implies the truth of another formula

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*

Logic in practice

1. **Define a logic**
   - Most often by researchers
   - Done once for all
2. **Choose the right logic** for the problem (L, I, $\models$)
3. **Write the theory**
4. **Different uses**
   - Basis for agreement developer/customer (theory)
   - Guarantee semantic interoperability (theory)
   - Ascertain that programs does what it is supposed to do (entailment)
   - Implement AI (entailment)

Trade off

There is a **trade-off** between

- **Expressive power** (expressiveness)
- **Computational efficiency** provided by a logical language

This trade-off is a measure of the **tension** between

- **Specification** expressiveness
- **Automation** efficiency

To use logic for **modeling**, the modeler must find the right trade off between expressiveness in the language for more tractable forms of reasoning

Example of expressiveness

Language	NL sentence	Formula
Propositional logic	Fausto likes skiing	Fausto-likes-skiing
	I like skiing	I-like-skiing
First-order logic	Every person likes skiing	∀ person.like-skiing(person)
	I like skiing	like-skiing(I)
	Fausto likes skiing	like-skiing(Fausto)
Modal logic	I believe I like skiing	Believe(I-like-skiing)
Description Logic	Every person likes cars	person ⊑ ∃ likes.Car

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

Computational complexity of reasoning (given decidability)

- The **difficulty to compute** a reasoning task in a given logic
- The logical languages are classified according to varying **degrees of complexity** (P, NP, Pspace, ...)

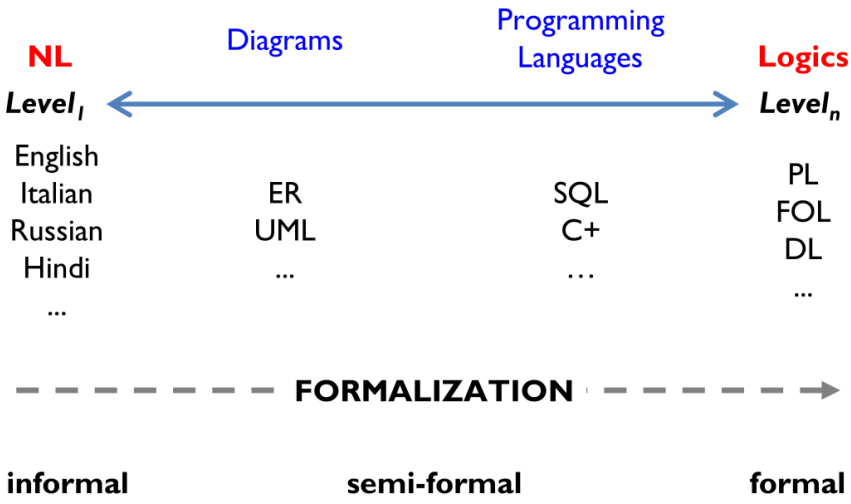
1.4. Formal and informal languages/models

Conceptual modeling

Specification models depends on the language

Model	Language	Description
Informal	Natural	Informal syntax and informal semantics
Semiformal	Diagrams, programming	Structured semi-formal languages with semi-formal syntax and informal semantics
Logic	Logical	Specific type of formal models

Levels of formalization



Uses of the languages

Language	Used for	Pros	Cons
Natural	Informal specification	Cheaper to use, direct, flexible, well-understood	Semantics is informal, ambiguous
	Beginning of problem solving	Interact with users	Pragmatically inefficient for automation
Diagrams	Semi-formal specification	Cheap to use	Semantics is informal, ambiguous
	Representation with unified languages, more precision	Structured and organized Interact with users	Pragmatically inefficient for automation
Logic	Formal specification	Well-understood, formal syntax and semantics	Hardly usable to interact with users
	Automation	Better to specify and prove correctness/completeness	Costly
	Reasoning services	Pragmatically efficient for automation exploiting the explicitly codified semantics	Effectiveness to be compared with Machine learning

2. Set theory

2.1. Introduction

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*

- **Extensional semantics of L** is based on the notion of "extension" of a formula (proposition) in L

2.2. Sets

Review slides T22

2.3. Relations

Basic concepts

> **Relation:** *a relation R from the set A to the set B is a subset of the Cartesian product of A and B*
>
> $$R \subseteq A \times B$$
>
> $$(x, y) \in R \;\Rightarrow\; xRy \;\; (x \text{ is } R\text{-related to } y)$$

- A **binary relation** on a set A is a subset $R \subseteq A \times A$

Given a relation R from A to B

- **Domain of R**

$$\mathrm{Dom}(R) = \{a \in A \mid \exists\, b \in B : aRb\}$$

- **Co-domain of R**

$$\mathrm{Cod}(R) = \{b \in B \mid \exists\, a \in A : aRb\}$$

- **Inverse relation of R**

$$R^{-1} \subseteq B \times A\,, \quad R^{-1} = \{(b, a) \mid (a, b) \in R\}$$

Properties

Let R be a binary relation on A, R is

- **Reflexive** $\iff aRa\,, \ \forall a \in A$
- **Symmetric** $\iff aRb \Rightarrow bRa\,, \ \forall a, b \in A$
- **Anti-symmetric** $\iff aRb \wedge bRa \Rightarrow a = b\,, \ \forall a, b \in A$
- **Transitive** $\iff aRb \wedge bRc \Rightarrow aRc\,, \ \forall a, b, c \in A$

Equivalence

Let R be a binary relation on A, R is an **equivalence relation** ($\sim$, $\equiv$) iff R is

- Reflexive
- Symmetric
- Transitive

Set partition

Let A be a set, a **partition** of A is a **family** F of non-empty subsets of A s.t.

- The subsets are ***pairwise*** [3] **disjoint**
- The **union** of all the subsets is the **set** A

Each element of A belongs to **exactly one** subset in F

Order relation

Let A be a set and R be a binary relation on A, R is a (partial) **order** $\leq$ if it is

- Reflexive $\quad a \leq a$
- Anti-symmetric $\quad a \leq b \land b \leq a \;\Rightarrow\; a = b$
- Transitive $\quad a \leq b \land b \leq c \;\Rightarrow\; a \leq c$

If the relation holds for all $a, b \in A$ then it is a **total order**

A relation is a **strict order** $<$ if it is

- Transitive $\quad a < b \land b < c \;\Rightarrow\; a < c$
- $\forall a, b \in A, \quad a < b \lor b < a \lor a = b$

2.4. Functions

Basic concepts

> **Function**: *given two sets A and B, a function f from A to B is a relation that associates to each element a in A exactly one element b in B*
>
> $$f : A \to B$$

- **Domain of f**

  $$\text{whole set } \; A$$

- **Image of a in A**

  $$b \in B \mid f(a) = b$$

- **Co-domain of f**

  $$\mathrm{Im}_f \subseteq B \;\mid\; \mathrm{Im}_f = \{b \in B \mid \exists\, a \in A : b = f(a)\}$$

Classes of functions

A function $f : A \to B$ is

- **Surjective** $\iff \exists\, a \in A \mid f(a) = b\,, \;\; \forall b \in B$

    - Each element in B is image of some elements in A
- **Injective** $\iff \exists! \, a \in A \mid f(a) = b, \ \forall b \in \text{Im}_f$
    - Distinct elements in A have distinct images in B
- **Bijective** $\iff \exists! \, a \in A \mid f(a) = b, \ \forall b \in B$
    - Injective and surjective

Inverse function

If $f : A \to B$ is **bijective** we can define its **inverse function**

$$f^{-1} : B \to A$$

For each function f we can define its **inverse relation**

- Such a relation is a **function** iff f is **bijective**

Composed function

Let $f : A \to B$ and $g : B \to C$ be functions, the **composition** of f and g is the function $g \circ f : A \to C$ obtained by applying f and then g

$$(g \circ f)(a) = g(f(a)) \quad \forall a \in A$$

$$g \circ f = \{(a, \, g(f(a))) \mid a \in A\}$$

3. Propositional logic

3.1. Intuition

Proposition

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

- A proposition can be **true** or **false**
- The same proposition can be expressed in **different ways**
- The **language** of PL allows us to express propositions
- Propositions are the **atomic language element** of PL
- PL is the **logic of propositions**

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

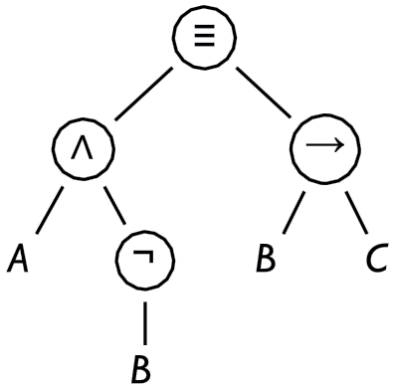
> Definitions (**Constants and variables**)
>
> - **Constant**: simple proposition
> - **Variable**: proposition where a variable can be substituted with a constant, a variable or in general any formula

Formulas as trees

A formula can be seen as a **tree**

- Leaves → **variables**
- Nodes → **connectives**

$$(A \wedge \neg B) \equiv (B \to C)$$



Subformulas

> Definition (**Subformulas**)
>
> - A SFof A
> - A SFof $\neg A$
> - A, B SFof $\{A \wedge B,\ A \vee B,\ A \to B,\ A \equiv B\}$
> - A SFof $B \wedge B$ SFof $C \Rightarrow A$ SFof C
>
> A is a **proper subformula** of B if A is a subformula of B and $A \neq B$

The **subformulas** of a formula represented as a tree correspond to all the different **subtrees** of the tree associated to the formula, one for each node

- Every formula has a **finite** number of subformulas

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

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

- Given the **cardinality** $|\texttt{PROP}|$, then there are $2^{|\texttt{PROP}|}$ different interpretations
  - All the different subsets (**power set**) of $\texttt{PROP}$
  - If $|\texttt{PROP}|$ is finite then there is a **finite** number of interpretations

Examples

1. $P = \{P, Q\} \quad | \quad I(P) = \top \land I(Q) = \bot \quad \Rightarrow \quad I = \{P\}$

2.	p	q	r	Set
I_1	$\top$	$\top$	$\top$	$\{p, q, r\}$
I_2	$\top$	$\top$	$\bot$	$\{p, q\}$
I_3	$\top$	$\bot$	$\top$	$\{p, r\}$
I_4	$\top$	$\bot$	$\bot$	$\{p\}$
I_5	$\bot$	$\top$	$\top$	$\{q, r\}$
I_6	$\bot$	$\top$	$\bot$	$\{q\}$
I_7	$\bot$	$\bot$	$\top$	$\{r\}$
I_8	$\bot$	$\bot$	$\bot$	$\{\}$

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

- $I(P) = \top \quad \Rightarrow \quad I \models P, \quad P \in \texttt{PROP}$

- $\neg(I \models A) \quad \Rightarrow \quad I \models \neg A$
- $I \models A \quad \wedge \quad I \models B \quad \Rightarrow \quad I \models A \wedge B$
- $I \models A \quad \vee \quad I \models B \quad \Rightarrow \quad I \models A \vee B$
- $I \models A \quad \rightarrow \quad I \models B \quad \Rightarrow \quad I \models A \rightarrow B$
- $I \models A \iff I \models B \quad \Rightarrow \quad I \models A \equiv B$

- $\forall\, P \in A,\ \ I(P) = I'(P) \ \Rightarrow\ I \models A \iff I' \models A$

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

Example slide 6

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**)

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

A		$\neg A$
VAL	$\to$	UNSAT
SAT	$\to$	$\neg\text{VAL}$
$\neg\text{VAL}$	$\to$	SAT
UNSAT	$\to$	VAL

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

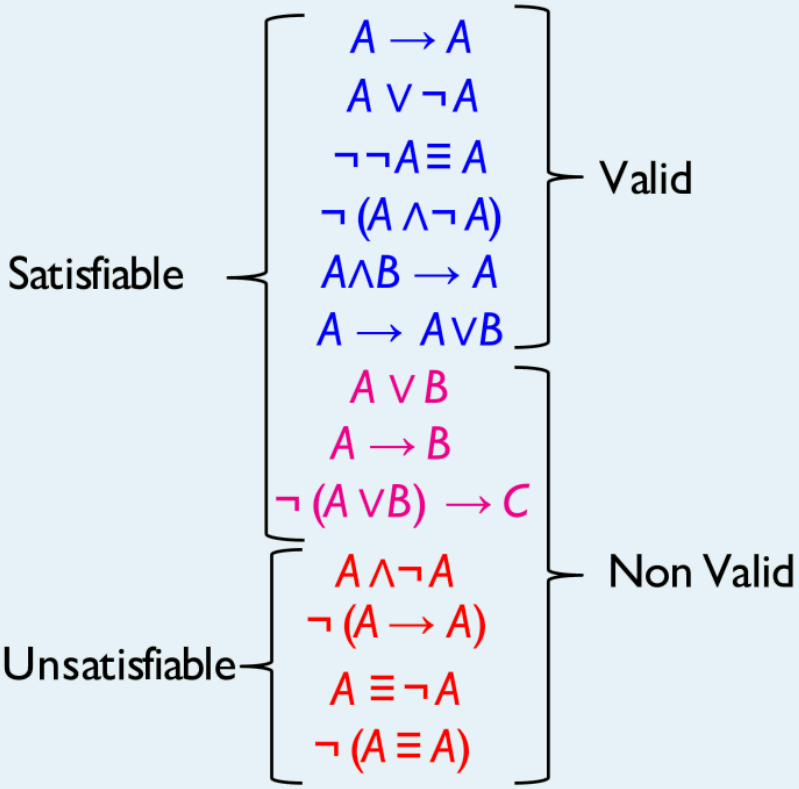
Theorem (**Refutation principle - validity**) (Please tell me if it is correct)

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

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
- $\Rightarrow \quad \not\models \neg A \quad \Rightarrow \quad \models A$

Examples



Sets of formulas

Definition (**Set of formulas**)

A **set of formulas** Γ is

- **Valid** $\iff \forall A \in \Gamma,\ \forall I,\ \ I \models A$
- **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**)

$$\forall_{<\infty}\,\Gamma = \{A_1, \ldots, A_n\}, \quad \Gamma \text{ valid} \iff A_1 \wedge \cdots \wedge A_n \text{ valid}$$
(resp. satisfiable and unsatisfiable)

Truth Table

Checking (un)**satisfiability** and **validity** of a formula A can be done by **enumerating** all the **interpretations** which are relevant for S, and for each interpretation I check if $I \models A$

Examples slides 5, 8-11

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*

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

Example slide 3-4, 6

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

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**

Examples slides 3-4

Axiomatization

> **Set of axioms for a theory**: *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 $cl(\Gamma)$

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

Hilbert axioms for PL

Axioms

1. $\varphi \to (\psi \to \varphi)$
2. $(\varphi \to (\psi \to \theta)) \to ((\varphi \to \psi) \to (\varphi \to \theta))$
3. $(\neg\psi \to \neg\varphi) \to ((\neg\psi \to \varphi) \to \varphi)$

Notes (**Go to Hilbert calculus**)

1. **Minimal language** (logical connectives defined in terms of the basic ones)
2. It allows to compute all **tautologies**
3. Useful to prove **properties** of logical theories
4. **Never** used **in practice** in CS
5. **In practice** in CS, people add **more axioms** (facts) which define what is true in the intended (mental) model

Compactness and minimality

The **axiomatization of a theory** is a **compact** way to represent

- A set of **interpretations**
- A set of possible (acceptable) **world states**
- All the **knowledge** we have of the real world

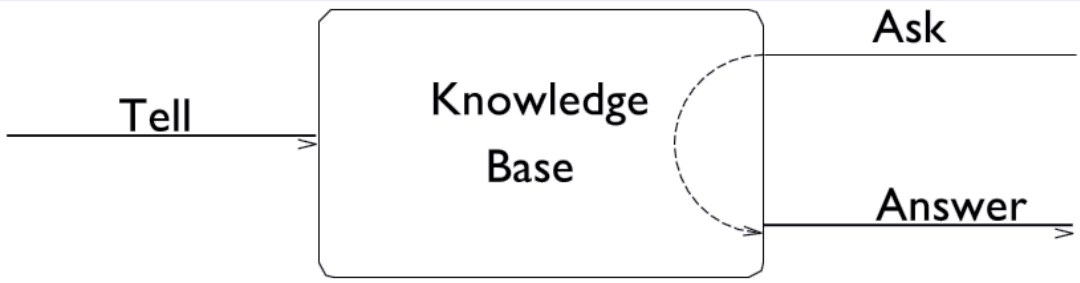
Sets of axioms are **minimal**

- **No** axioms can be **derived** from the others
- The **axioms** of a theory constitute the **basic knowledge**

- All the **knowledge** can be obtained by **logical consequence**

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* [4] 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

Examples slides 3-8

4.2. Decision problems

Reasoning / Decision problems

Four types of **questions**

- **Model checking**

  What is the truth value of φ in I ?

  $$\mathrm{MC}(I, \varphi) \ : \ I \overset{?}{\models} \varphi$$

- (Un)**Satisfiability**

  Is there a model I that satisfies φ ?

  $$\mathrm{SAT/UNSAT}(\varphi) \ : \ \exists I \mid I \overset{?}{\models} \varphi$$

- **Validity**

  Is φ satisfied by all the models I ?

  $$\mathrm{VAL}(\varphi) \ : \ \forall I, \ I \overset{?}{\models} \varphi$$

- **Logical consequence**

  Is φ satisfied by all the models I that satisfy all the formulas in Γ ?

  $$\mathrm{LC}(\Gamma, \varphi) \ : \ \Gamma \overset{?}{\models} \varphi$$

Model checking

Definition (**MC decision procedure**)

Algorithm that checks if a **formula** φ is satisfied by an **interpretation** I

$$\begin{aligned} \mathrm{MCDP}(\varphi, I) &= \top \iff I \models \varphi \\ \mathrm{MCDP}(\varphi, I) &= \bot \iff I \not\models \varphi \end{aligned}$$

- Returns for all inputs **either true or false**

Algorithm (**MCDP naive**)

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

true	$\wedge$	true	=	true	true	$\rightarrow$	true	=	true
true	$\wedge$	false	=	false	true	$\rightarrow$	false	=	false
false	$\wedge$	true	=	false	false	$\rightarrow$	true	=	true
false	$\wedge$	false	=	false	false	$\rightarrow$	false	=	true
true	$\vee$	true	=	true	true	$\equiv$	true	=	true
true	$\vee$	false	=	true	true	$\equiv$	false	=	false
false	$\vee$	true	=	true	false	$\equiv$	true	=	false
false	$\vee$	false	=	false	false	$\equiv$	false	=	true
	$\neg$	true	=	false					
	$\neg$	false	=	true					

Example slide 6

Algorithm (**MCDP lazy evaluation**)

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

- $A = p$

```
bool MCDP(I, p)
    if I(p) == true
        return YES
    else
        return NO
```

- … 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 [5] 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, in the shape*

$$(I_{11} \vee \cdots \vee I_{1n_1}) \wedge \cdots \wedge (I_{m1} \vee \cdots \vee I_{mn_m}) \equiv \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{n_j} I_{ij} \right)$$

- I_{ij} is the j-th literal of the i-th clause composing φ

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
  - Clauses with **different order** of literals are **equivalent**
- Multiple literals can be **merged**
  - Clauses with **different number** (≥ 1) of the same literal are **equivalent**
- Clauses as **set of literals**
  - A clause could be **represented** by a set of literals
  - Disjunctions and replications left **implicit**

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*

$$
\begin{aligned}
\mathrm{CNF}(p) &= p \\
\mathrm{CNF}(\neg p) &= \neg p \\
\mathrm{CNF}(\varphi \to \psi) &= \mathrm{CNF}(\neg\varphi) \otimes \mathrm{CNF}(\psi) \\
\mathrm{CNF}(\varphi \wedge \psi) &= \mathrm{CNF}(\varphi) \wedge \mathrm{CNF}(\psi) \\
\mathrm{CNF}(\varphi \vee \psi) &= \mathrm{CNF}(\varphi) \otimes \mathrm{CNF}(\psi) \\
\mathrm{CNF}(\varphi \equiv \psi) &= \mathrm{CNF}(\varphi \to \psi) \wedge \mathrm{CNF}(\psi \to \varphi) \\
\mathrm{CNF}(\neg\neg\varphi) &= \mathrm{CNF}(\varphi) \\
\mathrm{CNF}(\neg(\varphi \to \psi)) &= \mathrm{CNF}(\varphi) \wedge \mathrm{CNF}(\neg\psi) \\
\mathrm{CNF}(\neg(\varphi \wedge \psi)) &= \mathrm{CNF}(\neg\varphi) \otimes \mathrm{CNF}(\neg\psi) \\
\mathrm{CNF}(\neg(\varphi \vee \psi)) &= \mathrm{CNF}(\neg\varphi) \wedge \mathrm{CNF}(\neg\psi) \\
\mathrm{CNF}(\neg(\varphi \equiv \psi)) &= \mathrm{CNF}(\varphi \wedge \neg\psi) \otimes \mathrm{CNF}(\psi \wedge \neg\varphi)
\end{aligned}
$$

- $p \in \texttt{PROP}$
- $(C_1 \wedge \cdots \wedge C_n) \otimes (D_1 \wedge \cdots \wedge D_m) :=$
  $:= (C_1 \vee D_1) \wedge \cdots \wedge (C_1 \vee D_m) \wedge \cdots \wedge (C_n \vee D_1) \wedge \cdots \wedge (C_n \vee D_m)$

Examples slides 10-12

Pros and cons

Pros

- Simpler since it uses only **3 connective** in a very specific form
- **Checking satisfiability/validity** of a formula in CNF is easier

Cons

- CNF reduction could **grow exponentially**

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 if a model I satisfies N we do **not** need to know the **truth values** that I assigns to **all the literals** appearing in N

$N = \{C_0, \dots, C_n\} = \mathrm{CNF}(\varphi)$

- $I \models \varphi \iff \forall i = 0, \dots, n\,,\ I \models C_i$
- $I \models C_i \iff \exists l \in C_i \mid I \models l$

Partial valuation

Procedure (**DPLL**)

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

Under a **partial valuation** I, literals and clauses can be

- **True**

$$C = \top \iff \exists l \in C \mid l = \top$$

- **False** (conflicting)

$$C = \bot \iff \forall l \in C,\ l = \bot$$

- **Undefined** (unresolved)

$$C = \ ? \iff C \neq \top \wedge C \neq \bot$$

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 $C = \{l\}$*

- $\models \varphi = \{\{l\}\} \iff l = \top$

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
```

- **UP is enough** to decide the satisfiability when it terminates with this results
  - $\{\}$ $\Rightarrow$ satisfiable
  - $\{\{\}, \cdots\}$ $\Rightarrow$ unsatisfiable
- **UP does not terminate** (we have to guess) 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

Reasoning / Decision problems

Model Checking (see above) not via 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** node(s) is **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** ($\Leftarrow$)

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

Examples slide 7-8

Types of deductions

Two **types** of deductions as defined by **different logic**

	Forward deductions	Backward deductions
Defined by	Logic with forward calculus	Logic with backward calculus
Generate	Theorems from theorems	Sub-goals from goals
Good for	Proving properties of logic	Reasoning
Used in	Mathematical logic	CS / AI
Premises	What is known or assumed (axioms or assumptions)	The goal to be proved
Conclusions	What we want to discover (theorems/goals)	Termination condition which guarantees that the goal derives from what is known
Shape	Forward path/tree/DAG	Backward DAG
Problem	How do you know where to go? Search motivated by goal	In which direction to expand the proof, given exponential blow up Need very complex heuristics

5.2. Hilbert systems (VAL - forward chaining)

Hilbert calculus

$\Gamma \vdash A$

- Axioms
  1. $\varphi \to (\psi \to \varphi)$
  2. $(\varphi \to (\psi \to \theta)) \to ((\varphi \to \psi) \to (\varphi \to \theta))$
  3. $(\neg\psi \to \neg\varphi) \to ((\neg\psi \to \varphi) \to \varphi)$
- Inference rule: MP (modus ponens)

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

- Assumptions: Γ
- Theorems: A
- Deduction: as sequence
- Correctness and completeness theorems: $\Gamma \vdash \varphi \iff \Gamma \models \varphi$

Notes

Example of deduction

- Let L be a **PL** with $\texttt{PROP} = \{A, B, C\}$
- Let $T = \{A, (A \to B), (B \to C)\}$ **theory** (set of axioms)
  - Represents intended **mental model**
- Let C be the **theorem** that we want **to prove**
  - Namely we want to **prove** $T \vdash C$

Proof / deduction

1. A (hypothesis)
2. $(A \to B)$ (hypothesis)
3. B (2 applied to 1, via MP)
4. $(B \to C)$ (hypothesis)
5. C (4 applied to 3, via MP)

Linear representation of deduction

Automatic reasoning based on Hilbert Style

- HS proof system was invented with the main purpose of describing the **minimal rational assumptions** behind mathematical reasoning
- HS proofs are supposed to be provided by **humans**, who can use their intuition to apply smart heuristics to generate them
- Writing an algorithm that decides on the **validity of a formula** by searching a HS proof, is not a good idea
    - We look at **alternative ways** to write algorithms for deciding the validity of a FOL formula

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 exhaustively builds a branch for **any possible truth assignment**
    - **Interpretation** corresponding to a **branch** should **satisfy all the formulas** that appear in the branch
- Sometimes it is **not possible** to construct the tableaux since the **model** of the formula is **infinite**
- The basic idea is to **incrementally build the model** by
    - **Looking** at the formula
    - **Decomposing** it in a top/down fashion

> **Tableau method**: *method for proving, in a mechanical manner, that a given set of formulas is not satisfiable*

In particular, this 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

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

$\neg\neg$ 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

$$\frac{\begin{array}{l}\varphi\\ \neg\varphi\end{array}}{\times}$$

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

Smullyan's uniform notation

Two **types of formulas**

- **Conjunctive** (α)
  - Deterministic rules
  - If a model satisfies a conjunction, then it satisfies each of them
- **Disjunctive** (β)
  - Splitting rules
  - If a model satisfies a disjunction, then it satisfies one of them

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 $\Gamma \cup \{\neg\varphi\}$*

> **Fairness**: *a tableaux is fair if every non-literal of a branch gets eventually analyzed on this branch*

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
$\neg$SAT	VAL	SAT	$\neg$VAL		
$\downarrow$	$\downarrow$	$\wedge$	$\wedge$	SAT	$\neg$VAL
$\neg$VAL	SAT	$\neg$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**

Example slides 15-17

Termination

> Theorem (**Termination**)
>
> For any propositional tableau, after a **finite number of steps** no more expansion rules will be applicable
>
> Hint for proof
>
> Each rule results in ever shorter formulas

- Termination will not hold in the **first-order** case

Soundness and completeness

> Theorem (**Soundness** [6])
>
> $$\Gamma \vdash \varphi \;\Rightarrow\; \Gamma \models \varphi$$

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

- $\Gamma \vdash \varphi$ means that there exists a closed tableau for $\Gamma \cup \{\neg\varphi\}$

Decidability

> Theorem (**Decidability**)
>
> Tableau method is a **decision procedure** for classical propositional logic
>
> 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**)

Strategies of expansion

Using the wrong policy (expanding disjunctions first) leads to an **increase of size** of the tableau, which leads to an increase of time

- **Unsatisfiability** is still proved if set is unsatisfiable
- This is not the case for **other logic**, where applying the wrong policy may inhibit proving unsatisfiability of some unsatisfiable sets

Finding short proofs

Open problem: find an efficient algorithm to decide in all cases which rule to use next in order to derive the shortest possible proof

Guideline: always apply any applicable non-branching rules first

- In some cases, these may turn out to be **redundant**, but they will never cause an exponential blow-up of the proof

Efficiency

- **Truth-tables**: exponential in number of atoms
  - To check a formula involving n atoms requires filling in 2^n rows
- **Tableaux**: in the worst case exponential
  - Might skip some of the 2^n rows

7. First order logic

7.1. Intuition

Expressiveness of propositional logic

Try to express **statements in PL** through **atomic propositions**

- **Generalization patterns** are difficult to express
  - Universality
  - Independence

First order logic (like natural language) provides a way

- Of **directly representing facts**
- To **infer** the statements

Example slides 3-8

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

$$[\text{Qt}]\ x.\ \text{Proposition}(x)$$

- **Propositional logic** assumes world contains **facts**

Examples slides 9-12

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** [7]
  - a set $f_1, f_2, \ldots$ of **functional symbols** [8]
  - a set $P_1, P_2, \ldots$ of **relational symbols** [8]

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

Examples slides 4-6

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

- $t_1, \dots, t_n$ terms, P_i relational $\mid \#P_i = n \Rightarrow P_i(t_1, \dots, t_n)$ 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

Examples slide 8

Representation

Examples slides 9-10

Common mistakes

- **Use of $\wedge$ with $\forall$**
  - $\forall x \left(\mathrm{WorksAt}(\mathrm{FBK}, x) \wedge \mathrm{Smart}(x)\right)$ ❌

    "Everyone works at FBK and everyone is smart"
  - $\forall x \left(\mathrm{WorksAt}(\mathrm{FBK}, x) \to \mathrm{Smart}(x)\right)$ ✔

    "Everyone working at FBK is smart"
- **Use of $\to$ with $\exists$**
  - $\exists x \left(\mathrm{WorksAt}(\mathrm{FBK}, x) \to \mathrm{Smart}(x)\right)$ ❌

    "There is a person so that if (s)he works at FBK then (s)he is smart"

    This is true as soon as there is at least an x who does not work at FBK
  - $\exists x \left(\mathrm{WorksAt}(\mathrm{FBK}, x) \wedge \mathrm{Smart}(x)\right)$ ✔

    "There is an FBK-working smart person"

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**)

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

Examples slides 12-13

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)
  - $I(f_i) : \Delta^n \rightarrow \Delta$ (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.* [9] 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_k is free in $P(t_1, \ldots, t_k, \ldots, t_n)$
- 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*

Examples slides 4-6

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
- This equivalence with **closed formulas** holds for **all assignments**

Properties of quantifiers

The following formulas are **valid**

- $\forall x(\varphi(x) \wedge \psi(x)) \equiv \forall x\varphi(x) \wedge \forall x\psi(x)$
- $\exists x(\varphi(x) \vee \psi(x)) \equiv \exists x\varphi(x) \vee \exists x\psi(x)$
- $\forall x\varphi(x) \equiv \neg\exists x\neg\varphi(x)$
- $\forall x\exists x\varphi(x) \equiv \exists x\varphi(x)$
- $\exists x\forall x\varphi(x) \equiv \forall x\varphi(x)$

The following formulas are **not valid**

- $\forall x(\varphi(x) \vee \psi(x)) \equiv \forall x\varphi(x) \vee \forall x\psi(x)$

- $\exists x(\varphi(x) \wedge \psi(x)) \equiv \exists x\varphi(x) \wedge \exists x\psi(x)$
- $\forall x\varphi(x) \equiv \exists x\varphi(x)$
- $\forall x\exists y\varphi(x,y) \equiv \exists y\forall x\varphi(x,y)$

7.6. Exercises

See pdf 76

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*
>
> $$\text{UNA} := \varphi_{\Delta=\{c_1,\ldots,c_n\}} = \left(\bigwedge_{i\neq j=1}^{n} c_i \neq c_j \wedge \forall x\left(\bigvee_{i=1}^{n} x = c_i\right)\right)$$

- The language contains the constants $c_1, \ldots, c_n$, 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

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

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

Generalizing

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

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

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** $\varphi(x_1,\dots,x_n)$ with free variables $x_1,\dots,x_n$

> Definition (**Query answering**)
>
> Given
>
> - **Interpretation** I (a database instance) of a FOL language
> - **Formula** $\varphi(x_1,\dots,x_n)$ with n free variables
>
> Find all the n-**tuples** of elements of domain $(d_1,\dots,d_n) \in (\Delta^I)^n$ 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 [10]
Domain Δ [11]	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 [12]	Answers

8. FOL - reasoning as deduction

8.1. Reasoning problems

See PDF 81 for recap

8.2. Hilbert systems (VAL - forward chaining)

Hilbert style axiomatization

- Axioms for **propositional connectives**
  1. $\varphi \to (\psi \to \varphi)$
  2. $(\varphi \to (\psi \to \theta)) \to ((\varphi \to \psi) \to (\varphi \to \theta))$
  3. $(\neg\psi \to \neg\varphi) \to ((\neg\psi \to \varphi) \to \varphi)$
  - **MP** (modus ponens) $\dfrac{\varphi \qquad \varphi \to \psi}{\psi}$
- Axioms and rules for **quantifiers**
  4. $\forall x.\,(\varphi(x)) \to \varphi(t)$ if t is free for x in $\varphi(x)$
  5. $\forall x.\,(\varphi \to \psi) \to (\varphi \to \forall x.\,\psi)$ if x does not occur free in φ
  - **Gen** $\dfrac{\varphi}{\forall x.\,\varphi}$

Hilbert FOL is correct and complete with respect of **first order semantics**

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**

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

β **rules**

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

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

γ **rules**

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

δ **rules**

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

  - 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

Example slide 7

Universal quantification rule

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

- $\forall x.\, \varphi(x)$ means that **for every object** of the domain, the **property** $\varphi(x)$ should be **true**

- A **term** t that occurs in the tableaux denotes an **object of the domain**
- Therefore, $\varphi(t)$ must be **true for all the terms** t that occurs in the tableaux
    - The $\forall$ rule can be applied as **many times** as one want to **any term** that appears in the tableaux

Existential quantification rule

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

- $\exists x.\, \varphi(x)$ means that **for some object** of the domain, $\varphi(x)$ should be **true**
- But we **don't know which object** of the domain has the property φ
- Therefore, this rule cannot be applied to the **terms that already occur** in the tableaux, since otherwise we would introduce an **unjustified property**
- The trick is to introduce **fresh constant**
- Therefore, we allow only to **infer** $\varphi(c)$ form $\exists x.\, \varphi(x)$, where c is **fresh**

8.4. Examples

Examples PDF 82 slides 13-16

Examples PDF 84

8.5. Termination

Problem of non termination

For certain formulas there is the possibility of **infinite branches**

Key role of **function symbols** as **generators of** an unbound number of **terms**

Example slide 3

Infinite domains

In FOL **models** can be **infinite**

There are formulas which are **satisfied** only by **infinite models**

Examples slides 4-5

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*
>
> $$\forall x \varphi \qquad \neg \exists x \varphi$$

> 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
- Only for **special cases**
  - A single **1-place function symbol** together with a **constant** is already enough to construct **infinitely many terms**

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 we can construct using the function symbols appearing on the branch (Herbrand universe)
  - You can optionally introduce a **fake constant** for the value of the **term**
- **Function symbols**: interpreted as themselves (or using the fake constants)
- **Predicate symbols**: interpreted in terms of their occurrences in the branch

Example slide 5

1. implicazione ↩

2. conformità ↩

3. a coppie ↩

4. per mezzo di ↩

5. sfruttando ↩

6. solidità (correctness) ↩

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

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

9. with respect to ↩

10. (tables) ↩

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

12. (queries) ↩