Advanced Data Management

First-Order Logic

Domenico Fabio Savo

Corso di laurea magistrale
INGEGNERIA INFORMATICA
Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione
University of Bergamo

Outline of the course

1. Introduction to propositional logic
2. **Introduction to First-order Logic**
3. Relational Calculus
4. **I**nformation **I**ntegration **S**ystems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. **O**ntology-**B**ased **D**ata **M**anagement (OBDM)
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

Propositional Logic (Recap)

Exercise 0 (our friends Ander and Bjorn)

On a walk in the woods, you stumble upon a bridge guarded by two trolls. They say:

Ander: If I wear a tie or cufflinks then I wear a jacket.

Bjorn: Ander is not wearing a jacket!

Prove or disprove that:

1. Ander wears cufflinks.
2. Ander wears a tie.

Exercise 0: Solution

On a walk in the woods, you stumble upon a bridge guarded by two trolls. They say:

Ander: If I wear a tie or cufflinks then I wear a jacket.

Bjorn: Ander is not wearing a jacket!

- Let C = Cufflink, T = Tie, J = Jacket
- Consider the theory $\{\neg(C \vee T) \vee J, \neg J\}$
- In all the models I of the theory above we have that $I(J) = \boldsymbol{false}$
- Therefore, $I(\neg(C \vee T)) = \boldsymbol{true}$.
- **By De Morgan:** $\neg(C \vee T) \equiv (\neg C \wedge \neg T)$

Predicate Logics: An Example

The Need For Predicates

With propositional logic, we can formally reason about the truth or falsity of a sentence starting from the truth or falsity of some basic facts

- Basic facts are represented by propositional variables
- Logical connectives are used to represent how the facts are connected

Often (especially in databases) we need to talk about **properties of sets**

- Instead of propositional variables we use **constants** and **predicates**, i.e., *syntactic representations* of **individuals** and of **sets of individuals**, respectively.

Example of Predicates

Suppose you want to formalize the fact that Fabio is a person that lives in a city called Bergamo (the subscript indicates the *arity* of the predicate).

- **Predicates**: $\text{Person}_{/1}$, $\text{City}_{/1}$, $\text{Lives}_{/2}$.
- **Constants:** Fabio, Bergamo

$\text{Person}(\text{Fabio}) \wedge \text{City}(\text{Bergamo}) \wedge \text{Lives}(\text{Fabio}, \text{Bergamo})$

  – This formula could capture our intuition.

Example of Predicates

Clearly one could write different sentences using the same predicates.

- **Predicates**: $\text{Person}_{/1}$, $\text{City}_{/1}$, $\text{Lives}_{/2}$.

Person(Bergamo)

– Bergamo is a Person.

City(Pavia) ∧ Lives(Fabio, Pavia)

– Fabio lives in Pavia.

Example of Functions

We may also want to talk about objects we do not explicitly know. In this case we use functions.

- **Predicates**: $\text{Person}_{/1}$, $\text{City}_{/1}$, $\text{Lives}_{/2}$.
- **Constants:** Fabio, Bergamo
- **Functions**: $\text{FatherOf}_{/1}$,

Person(FatherOf(Fabio))

– The father of Fabio is a person.

City(Pavia) ∧ Lives(FatherOf(Fabio), Pavia)

– The father of Fabio lives in Pavia.

Truth or Falsity

Consider the following formula

– Person(Fabio) ∧ City(Bergamo) ∧ Lives(Fabio, Bergamo)

How can we define the truth value of such a formula?

Instead of evaluating each atom, we evaluate the value of the predicates on the objects represented by the constants and functions.

First-Order Logic

There are different formalisms to represent predicates.

First-order logic (FOL) is the logic to speak about objects.

FOL is concerned with properties of these objects and relations among them.

FOL uses functions (including constants) that denote objects.

First-Order Logic: Syntax

Syntax of First-Order Logic (1)

First-Order Logic formulae consists of the following components

1. **Terms**, intuitively representing individuals
2. **Predicates**, intuitively representing relations and properties of individuals
3. **Connectives**, intuitively representing the structure of the formula.
4. **Quantifiers**, intuitively representing individuals not explicitly occurring in the formula

Syntax of First-Order Logic (2)

In what follows we assume to have the following pairwise distinct countable sets:

- the set of symbols **Vars** $= \{x_1, x_2 \ldots, \}$
- the set of function symbols **Funcs**
  - A function symbol f has an associated *arity* ar(f) ($f_{/\mathrm{ar}(f)}$)
  - Functions of arity 0 are called **constants**.
- the set of predicate symbols **Pred**
  - A predicate symbol P has an associated *arity* ar(P) ($P_{/\mathrm{ar}(P)}$)

Note: a set is called *countable* if it is finite or countably infinite

Terms

Def: The set **Terms** is inductively defined as follows.

- **Vars** ⊆ **Terms**
- If $t_1, \ldots, t_n \in$ **Terms** and $f_{/n} \in$ **Funcs** then $f(t_1, \ldots, t_n) \in$ **Terms**.
- Nothing else is in **Terms**.

Further on this course, we will consider the fragment of FOL that includes only **constants** as functions (a **constant** is a function with arity 0).

In such case, we will simplify our formalization saying that the set **Terms** corresponds to **Cons** ∪ **Vars** where **Cons** is a countable set of constants symbols

Exercise

Assume **Vars** = $\{x_1, x_2\}$

Assume **Funcs** = $\{+_{/2}, *_{/2}, 1_{/0}\}$

Which of the following is in **Terms**?

- $+(1, x_1)$
- $*\,(+(1,1), 2)$
- $+(+(x_1, x_1), +(x_2, x_2))$
- $*\,(+(x_1, 1), +(1))$
- $*\,(+(x_1, x_2), +(x_3, x_4))$

Exercise: Solution

Assume **Vars** = $\{x_1, x_2\}$

Assume **Funcs** = $\{+_{/2}, *_{/2}, 1_{/0}\}$

Which of the following is in **Terms**?

- $+(1, x_1)$. **YES!**
- $*\,(+(1, 1), \mathbf{2})$. **NO!**
- $+(+(x_1, x_1), +(x_2, x_2))$. **YES!**
- $*\,(+(x_1, 1), +(\mathbf{1}))$. **NO!**
- $*\,(+(x_1, x_2), +(\boldsymbol{x_3}, \boldsymbol{x_4}))$. **NO!**

FOL Formulae

The set of **Forms** is defined inductively as follows:

If $t_1, \ldots, t_n \in$ **Terms** and $P_{/n} \in$ **Pred**, then $P(t_1, \ldots, t_n) \in$ **Forms**

If $t_1, t_2 \in$ **Terms** then $(t_1 = t_2) \in$ **Forms**

} These are called ***atomic formulae***

If $f_1, f_2 \in$ **Forms** and $x \in$ **Vars** then

- $(f_1 \wedge f_2) \in$ **Forms**
- $(f_1 \vee f_2) \in$ **Forms**
- $(f_1 \rightarrow f_2) \in$ **Forms**
- $\neg(f_1) \in$ **Forms**
- $\exists x. f_1 \in$ **Forms**
- $\forall x. f_1 \in$ **Forms.**

Nothing else is in **Forms**

Functions and Predicates

The sets **Preds**, **Funcs**, and **Vars** form the **alphabet** of our logic.

At the syntactic level predicates and functions look similar

- There is however a difference in the syntactic rules that define them

The difference will be clear at the semantic level

- Functions define individuals
- Predicates define relations among individuals

Propositional and Predicate Logics

- Connectives work the same in both logics
  - At the syntactic level

- A predicate P with ar(0) can be seen as propositional variables.
  - First-Order Logic is an extension of Propositional Logic

- In addition we can talk about objects:
  - Constants and variables representing specific objects
  - $\exists x$ **existential quantification**: there exists an object such that.
  - $\forall x$ **universal quantification**: all objects such that.

Exercise

Assume the following alphabet:

Vars = {x, y}, **Funcs** = {$+_{/2}$, $*_{/2}$, $1_{/0,}$ $2_{/0}$, …}, **Preds** = {$\text{Even}_{/1}$, $\text{Odd}_{/1}$}

Which of the following are FOL formulae?

1. $Odd(2)$
2. $\neg\, Odd(2) \vee \neg\, Nat(1)$
3. $\neg \vee\ Odd(3)$
4. $\forall x. \exists y. Odd(+(x, y)) \wedge Nat(x) \wedge Nat(y)$

Exercise: Solution

Assume the following alphabet:

Vars = {x, y}, **Funcs** = {$+_{/2}$, $*_{/2}$, $1_{/0,}$ $2_{/0}$, …}, **Preds** = {$\text{Even}_{/1}$, $\text{Odd}_{/1}$}

Which of the following are FOL formulae?

1. $Odd(2)$ **YES!**
2. $\neg\, Odd(2) \vee \neg\, Nat(1)$ **YES!**
3. $\neg \vee\ Odd(3)$ **NO!**
4. $\forall x. \exists y. Odd(+(x, y)) \wedge Nat(x) \wedge Nat(y)$ **YES!**

Exercise: observation

Which of the following are FOL formulae?

1. $Odd(2)$ **YES!**
2. $\neg\, Odd(2) \vee \neg\, Nat(1)$ **YES!**
3. $\neg \vee\ Odd(3)$ **NO!**
4. $\forall x.\exists y.Odd(+(x,y)) \wedge Nat(x) \wedge Nat(y)$ **YES!**

Can we establish whether the above formulae are true?

Exercise: observation

Which of the following are FOL formulae?

1. $Odd(2)$ **YES!**
2. $\neg\, Odd(2) \vee \neg\, Nat(1)$ **YES!**
3. $\neg \vee\ Odd(3)$ **NO!**
4. $\forall x.\exists y.Odd(+(x,y)) \wedge Nat(x) \wedge Nat(y)$ **YES!**

Can we establish whether the above formulae are true?

Not yet! For now, formulae are meaningless syntactic objects.

To give meaning to the symbols we need **interpretations** that, similarly to Propositional logic, will tell us whether a formula is **true** of **false**.

First-Order Logic: Semantics

Domains and Interpretations

In order to interpret FOL formulae, we need a set of objects.

We will call this set **domain of discourse** (or simply domain).

- The domain of natural numbers …
- The domain of people and their jobs …

Observe: objects (i.e., elements in the domain) and terms (elements of the set **Terms** define earlier) are not the same!

- Function $2_{/0}$ is not the number 2 in the domain of natural numbers.

To connect terms with objects we use **interpretations**.

- Intuitively, interpretations connect terms to their "meaning"

Relations

Before providing the notion of interpretation, we need the following notion of **relation**

Assume to have a set $\mathcal{A}$ of elements

An **n-ary tuple** over $\mathcal{A}$ is a sequence of **n** elements of $\mathcal{A}$

- Example: $(a_1, a_2, \ldots, a_n)$

The **n-th power of $\mathcal{A}$** (written $\mathcal{A}^n$) is the set of **n-ary tuples** over $\mathcal{A}$

An **n-ary relation** over $\mathcal{A}$ is a **subset** of $\mathcal{A}^n$

FOL Interpretation: Definition

Assume sets **Funcs** and **Preds** of functions and predicates, respectively

Definition: An **interpretation** I is a pair $(\Delta^I, \ \cdot^I)$ where

- Δ^I is a countable set (the domain of discourse)
- $\cdot^I$ is a function from **Funcs** ∪ **Preds** defined as follows:
  - $f^I = \Delta^k \to \Delta$, for each $f \in$ **Funcs** with $ar(f) = k$
  - $P^I \subseteq \Delta^k$, for each $P \in$ **Preds** with $ar(P) = k$

So, the function $\cdot^I$ maps each function symbol f of arity n to an n-ary function $f^I = \Delta^k \to \Delta$ and each predicate symbol P of arity n to an n-ary relation $P^I \subseteq \Delta^k$.

In case for a function f we have that $ar(f) = 0$, then f^I denotes exactly one object in Δ^I. As already said, we call them **constants.**

Example of Interpretation

Assume the following:

Funcs = {$+_{/2}$, $*_{/2}$, $1_{/0}$, $2_{/0}$, …}, **Preds** = {$\text{Even}_{/1}$, $\text{Odd}_{/1}$}

In the standard interpretation I of natural numbers we have:

- Δ^I is equal to $\mathbb{N}$
- The predicate $\text{Even}_{/1}$ is mapped to the unary relation $Even^I \subseteq \Delta$, representing the set of even numbers
- The predicate $\text{Odd}_{/1}$ is mapped to the unary relation $Odd^I \subseteq \Delta$, representing the set of odd numbers
- Each costant $\text{n}_{/0}$ is mapped to a natural number: $n^I = n \in \mathbb{N}$

What is I telling us about the following formulae?

Interpretation of Formulae

Can we say whether the following formulae are **true** in I?

1. $Odd(2)$
2. $\neg\, Odd(2) \vee \neg\, Odd(1)$
3. $\exists x. Odd(x) \vee Even(x)$
4. $Odd(x) \vee Even(x)$

Interpretation of Formulae

Can we say whether the following formulae are **true** in I?

1. $Odd(2)$
2. $\neg\, Odd(2) \vee \neg\, Odd(1)$
3. $\exists x. Odd(x) \vee Even(x)$
4. $Odd(x) \vee Even(x)$

Intuitively, I tells everything we need to know about formulae 1 and 2. Moreover, by interpreting $\exists x$ as "there exists a domain element", we have that I still gives us the truth value of the formula 3.

Formula 4 is totally different: **we need a way to evaluate variables**.

Assignments

Assume an interpretation $I = (\Delta^I, \cdot^I)$ for **Funcs** and **Preds.**
An **assignment** $\boldsymbol{\alpha}$ is function that maps each variable symbol in **Vars** to an object in Δ^I, i.e.,

$$\alpha: \boldsymbol{Vars} \rightarrow \boldsymbol{\Delta^I}$$

We define the extension $\hat{\alpha}$ of α to **Terms** as follows.

- $\hat{\alpha}(x) = \alpha(x)$, for each $x \in$ **Vars.**
- $\hat{\alpha}(f(t_1, \ldots, tn)) = f^I\ (\hat{\alpha}(t_1), \ldots, \hat{\alpha}(t_n))$, for each $f \in$ **Terms.**

Given a variable $x \in$ **Vars** and an object $o \in \Delta^I$, we define the assignment $\alpha[x \rightarrow o]$ as

- $\alpha[x \rightarrow o](y) = \alpha(y)$, for each variable $y \neq x$
- $\alpha[x \rightarrow o](x) = o$

Truth of a Formula

Assume a FOL formula φ, an assignment α, and an interpretation I

We say that φ is true in I according to α (written $I, \alpha \vDash \varphi$) if the following holds:

- $I, \alpha \vDash P(x_1, \ldots, x_n)$ and $\left(\hat{\alpha}(x_1), \ldots, \hat{\alpha}(x_n)\right) \in P^I$
- $I, \alpha \vDash (x_1 = x_2)$ and $\hat{\alpha}(x_1) = \hat{\alpha}(x_2)$
- $I, \alpha \vDash (\varphi_1 \wedge \varphi_2)$ and $\mathrm{I}, \alpha \vDash \varphi_1$ and $\mathrm{I}, \alpha \vDash \varphi_2$
- $I, \alpha \vDash (\varphi_1 \vee \varphi_2)$ and $\mathrm{I}, \alpha \vDash \varphi_1$ or $\mathrm{I}, \alpha \vDash \varphi_2$
- $I, \alpha \vDash (\varphi_1 \rightarrow \varphi_2)$ and either $\mathrm{I}, \alpha \nvDash \varphi_1$ or $\mathrm{I}, \alpha \vDash \varphi_2$
- $I, \alpha \vDash \neg\varphi_1$ and $\mathrm{I}, \alpha \nvDash \varphi_1$ (φ is NOT true in I according to α)
- $I, \alpha \vDash \exists x.\, \varphi_1(x)$ and $\mathrm{I}, \alpha[x \rightarrow a] \vDash \varphi_1$, for some $a \in \Delta^I$
- $I, \alpha \vDash \forall x.\, \varphi_1(x)$ and $\mathrm{I}, \alpha[x \rightarrow a] \vDash \varphi_1$, for each $a \in \Delta^I$

Truth and Falsity

Assume a FOL formula φ.

- φ is called **satisfiable** if $I, \alpha \vDash \varphi$, for some I, α.
- φ is called **unsatisfiable** if $I, \alpha \vDash \neg\varphi$, for every I, α.
- φ is called **valid (tautology)** if $I, \alpha \vDash \varphi$, for every I, α.
- φ is called **falsifiable** if $I, \alpha \vDash \neg\varphi$, for some I, α.

Equalities – Symbols and Semantics

Given and interpretation I and an assignment α, we know that

$$I, \alpha \vDash (t_1 = t_2) \textbf{ if } \hat{\alpha}(t_1) = \hat{\alpha}(t_2)$$

What is the difference between the two equality symbols above?

- The = symbol on the left is **syntactic**, indeed, it represents a binary relation symbol with special interpretation.

- The = symbol on the right is **semantic**, it means **identity** over the domain of I

Exercise

Assume functions $t_{/0}$ and $s_{/0}$

Are the following formulae satisfiable, falsifiable, unsatisfiable, or valid?

1. $(t = t)$
2. $(t = s)$
3. $\neg(t = t)$
4. $\neg(t = s)$

Give examples of interpretations to support your claims

Exercise

Assume functions $t_{/0}$ and $s_{/0}$

Are the following formulae satisfiable, falsifiable, unsatisfiable, or valid?

1. $(t = t)$ **Valid (is a tautology)**
2. $(t = s)$ **Satisfiable, Falsifiable**
3. $\neg(t = t)$ **Unsatisfiable**
4. $\neg(t = s)$ **Satisfiable, Falsifiable**

Give examples of interpretations to support your claims

Truth Of a Formula: Additional Remarks

- Connectives work as in Propositional Logic.
  - Once we evaluate atoms, we essentially have the truth value of a propositional formula.

- To capture the intuitive meaning of quantification we use assignments.

Example of First-Order Formulae

Alphabet and Domain

- **Funcs** = {*Arethi*$_{/0}$, *Bob*$_{/0}$, *Dep1*$_{/0}$, *Dep2*$_{/0}$}
- **Preds** = {*Employee*$_{/1}$, *Department*$_{/1}$, *Works*$_{/2}$, *Directs*$_{/2}$}
- $\Delta^I = \{a, b, c, d1, d2\}$

- **Observe**: None of the constants are in Δ^I.

Some Questions

- Is *Dep1* a department or an employee?

- Can a constant be an object?

Some Questions

- Is *Dep1* a department or an employee?
  - Not necessarily. Depends on the interpretations.

- Can a constant be an object?
  - *Only if we assume to have the same object in the domain!*

Interpretation: Functions

- **Funcs** = {Arethi$_{/0}$, Bob$_{/0}$, Dep1$_{/0}$, Dep2$_{/0}$}
  - $\Delta^I = \{a, b, c, d1, d2\}$
  - Arethi$^I = a$
  - Bob$^I = b$
  - Dep1$^I = d1$
  - Dep2$^I = d2$

- **Observe:** we have no function for c!

Interpretation: Predicates

Preds = { $\text{Employee}_{/1}$, $\text{Department}_{/1}$, $\text{Works}_{/2}$, $\text{Directs}_{/2}$}

- $\Delta^I = \{a, b, c, d1, d2\}$
- $Employee^I = \{a, b, c\}$
- $Department^I = \{d1, d2\}$
- $Works^I = \{(a, d1), (b, d2), (c, d1)\}$
- $Directs^I = \{(a, d1), (c, d1), (c, d2)\}$

Observe: c takes part to the interpretation of predicates!

Exercise

Assume an assignment $\alpha(x) = a$.

Which of the following formulae are true in I, α ?

1. $(x = Arethi) \vee (x = Bob)$
2. $Employee(Arethi) \wedge Works(Arethi, Dep1)$
3. $\exists x.\, Directs(x, Dep2)$
4. $\forall x.\, Employee(x) \rightarrow \neg Department(x)$
5. $\forall x.\, \forall y.\, \forall z.\, \mathrm{Directs(x, y)} \wedge Directs(x, z) \rightarrow y = z$

Exercise: Solution

Assume the assignment $\alpha(x) = a$.

- $(x = Arethi) \vee (x = Bob)$
- $\hat{\alpha}(Arethi) = a$
- $\hat{\alpha}(Bob) = b$

We can conclude that the formula is true.

Observe, this depends on the assignment → Consider for instance a different assignment $\beta(x) = d1$

Exercise: Solution

Assume the assignment $\alpha(x) = a$.

- $Employee(Arethi) \wedge Works(Arethi, Dep1)$
- $\hat{\alpha}(Arethi) = a$
- $\hat{\alpha}(Dep1) = d1$
- $a \in Employee^I$ **and** $(a, d1) \in Works^I$

We can conclude that the formula is true.

Observe, this holds for every assignment (we have no variables)!

Exercise: Solution

Assume the assignment $\alpha(x) = a$.

- $\exists x.Directs(x, Dep2)$ is true if for some $o \in \Delta^I$ we have

$$I, \alpha[x = o] \vDash Directs(x, Dep2)$$

- Since $I, \alpha[x = c] \vDash Directs(x, Dep2)$, since we have that $\text{Dep2}^I = d2$ and $(c, d2) \in Directs^I$ **we can conclude that the formula is true.**

Observe: the original assignment for x does not really matter.

Exercise: Solution

Assume the assignment $\alpha(x) = a$.

- $\forall x. Employee(x) \rightarrow \neg Department(x)$ is true if for every $o \in \Delta^I$ we have:

$$I, \alpha[x = o] \vDash Employee(x) \rightarrow \neg Department(x)$$

- This is the case, therefore the formula is **true**.

Intuitively, the formula requires that the two sets are **disjoint**.

Exercise: Solution

Assume the assignment $\alpha(x) = a$.

- $\forall x.\forall y.\forall z.\mathrm{Directs}(\mathrm{x},\mathrm{y}) \wedge \mathrm{Directs}(x,z) \rightarrow y = z$ is true if for every $o, o', o'' \in \Delta^I$ we have:

$$I, \alpha[x = o][y = o'][z = o''] \vDash \mathrm{Directs}(\mathrm{x},\mathrm{y}) \wedge \mathrm{Directs}(x,z) \rightarrow y = z$$

- This is not the case for $\alpha[x = c][y = d1][z = d2]$. So, the formula is **false**!

Intuitively, the formula is a **key constraint** (see Database course)

Consequence and Equivalence

Implication and Equivalence

Let φ, ψ be two FOL formulae. We have that:

- φ **implies** ψ (written $\boldsymbol{\varphi \vDash \psi}$) if for every I, α s.t. $I, \alpha \vDash \varphi$ we have $I, \alpha \vDash \psi$
- φ **is equivalent to** ψ ($\boldsymbol{\varphi \equiv \psi}$) if for every I, α, we have that $I, \alpha \vDash \varphi$ if and only if $I, \alpha \vDash \psi$

We can extend **implication** and **equivalence** to **logical theories** (finite sets of logical formulae) in the natural way.

Useful Equivalences

Some useful equivalences for FOL Formulae

- **De Morgan's Law 1:** $\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi)$
- **De Morgan's Law 2:** $\neg(\varphi \vee \psi) \equiv (\neg\varphi \wedge \neg\psi)$
- **Double Negation:** $\neg\neg\varphi \equiv \varphi$
- **Negation of Existential:** $\neg\exists x.\varphi \equiv \forall x.\neg\varphi$
- **Negation of Universal:** $\neg\forall x.\varphi \equiv \exists x.\neg\varphi$

Exercise

Prove De Morgan’s Law 1 and the law of double negation.

Exercise: Solution

Prove De Morgan's Law 1 and the law of double negation.

De Morgan's Law 1: $\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi)$

Proof: We prove the claim showing the truth tables.

	Truth table $\neg(\varphi \wedge \psi)$	Truth table $(\neg\varphi \vee \neg\psi)$
• φ = **true,** ψ = **true.**	**false**	**false**
• φ = **true,** ψ = **false**	**true**	**true**
• φ = **false,** ψ = **true**	**true**	**true**
• φ = **false,** ψ = **false**	**true**	**true**

Exercise: Solution

Prove De Morgan's Law 1 and the law of double negation.

Double Negation: $\neg\neg\varphi \equiv \varphi$

Proof: We prove the claim showing the truth tables.

	Truth table φ	Truth table $\neg\neg\varphi$
• φ = **false**	**false**	**false**
• φ = **true**	**true**	**true**

Exercise

Prove Negation of Existential

$$\neg \exists x. \varphi \equiv \forall x. \neg \varphi$$

Exercise

Prove Negation of Existential

$$\neg\exists x.\varphi \equiv \forall x.\neg\varphi$$

Proof: If $I, \alpha \vDash \neg\exists x.\varphi$ then $I, \alpha \nvDash \exists x.\varphi$. Therefore, $I, \alpha[x = a] \vDash \neg\varphi$, for every $a \in \Delta^I$. In turn, this proves that $I, \alpha \vDash \forall x.\neg\varphi$.

If $I, \alpha \vDash \forall x.\neg\varphi$, then $I, \alpha[x = a] \vDash \neg\varphi$, for every $a \in \Delta^I$. In turn, this proves that $I, \alpha \nvDash \exists x.\varphi$.

Introduction to computational complexity

Computational complexity (1/2)

Computational complexity theory aims to study how difficult it is to solve specific problems.

Complexity theory deals with **decision problems**: i.e., problems that admit a **yes**/**no** answer.

A **decision algorithm** is an algorithm that computes the correct truth value for each input instance of a **decision problem (The algorithm has to terminate on all inputs)**:

- **input**: an instance of the problem
- **output**: **yes** or **no**

A decision problem is **decidable** if there exists a decision algorithm for it. Otherwise it is **undecidable**.

The complexity is measured in terms of the amount of **resources** (time, space) that the algorithm needs to solve the problem (complexity of the algorithm, or **upper bound**).

To measure the complexity of the problem, we consider the **best possible algorithm** that solves it (**lower bound**).

Computational complexity (2/2)

Worst-case complexity analysis: the complexity is measured in terms of a (complexity) function **f**:

- **argument**: the size n of an instance of the problem
- **result**: the amount **f**(n) of time/space needed in the worst-case to solve an instance of size n

To abstract away from contingent issues (e.g., programming language, processor speed, etc.), we refer to an abstract computing model: **Turing Machines** (TMs).

Usually one does not consider specific complexity functions f, but rather families **C** of complexity functions, giving rise to complexity classes.

Definition: **A time/space complexity class C** is the set of all problems P such that an instance of P of size n can be solved in time/space at most C(n).

Reductions

To establish **lower bounds** on the complexity of problems, we make use of the notion of reduction:

- **Definition**: A **reduction** from a problem P1 to a problem P2 is a function R from instances of P1 to instances of P2 such that:
  - R is efficiently computable (typically in logarithmic space), and
  - An instance I of P1 has answer yes if and only if R(I) has answer yes.

We say that P1 reduces to P2 if there is a reduction R from P1 to P2.

- **Intuition**: If P1 reduces to P2, then P2 is at least as difficult as P1, since we can solve an instance I of P1 by reducing it to the instance R(I) of P2 and then solve R(I).

Hardness and Completeness

If we can provide an algorithm that solve a problem P of size n by using at most C(n) time\space, than we can prove the **membership** of P to the class C (the **upper-bound**)

To provide a **lower-bound** we need to refer to the notion of **hardness**:

- **Definition**: A problem P is **hard** for a complexity class C if every problem in C can be reduced to P.

If we have both, we show the **completeness** w.r.t. a complexity class

- **Definition**: A problem P is **complete** for a complexity class C if it is **hard** for C, and it **belongs** to C (membership to C)

Intuitively, a problem that is complete for C is among the hardest problems in C.

Tractability and intractability: PTime and NP

Definition: **PTime** is the set of problems solvable in polynomial time by a **deterministic TM**.

- These problems are considered **tractable**, i.e., solvable for large inputs.

Definition: **NP** is the set of problems solvable in polynomial time by a **non-deterministic TM**.

- These problems are believed **intractable**, i.e., unsolvable for large inputs.
- The best known actual algorithms actually require exponential time.
- Corresponds to a large class of practical problems, for which the following type of algorithm can be used:
  1. Non-deterministically guess a possible solution of polynomial size.
  2. Check in polynomial time that the guessed solutions is good.

Complement of problems in NP: coNP

Definition: **coNP** is the set of problems whose complement is in NP, i.e., problems for which determining whether an instance admits a **no** answer is in **NP**.

For problems whose complexity is characterized in terms of a non-deterministic TM, solving the problem and solving its complement might be different.

The reason for this is that a **yes** answer is returned if there exists a non-deterministic computation-path of the TM that leads to acceptance. Instead, a **no** answer requires that all non-deterministic computation-paths of the TM lead to rejection.

Specifically, **coNP** is believed to be different from both **NP** and **PTime**.

Complexity classes above NP

Definition: **PSpace** is the set of problems solvable in polynomial space by a deterministic TM.

- Polynomial space is "**not really good**", since these problems may require exponential time. Indeed, these problems are believed to be more difficult than NP problems.

Definition: **ExpTime** is the set of problems solvable in exponential time by a deterministic TM.

- These problems are considered to be very difficult.

Definition: **NExpTime** is the set of problems solvable in exponential time by a non-deterministic TM.

Complexity classes below PTime

Definition: **LogSpace** (**NLogSpace)** is the set of problems solvable in logarithmic space by a (non-)deterministic TM.

- **Note**: when measuring the space complexity, the size of the input does not count, and only the working memory (TM tape) is considered.

Definition: **AC^0** is the set of problems solvable in constant time using a polynomial number of processors.

- These problems are solvable efficiently even for very large inputs.
- Corresponds to the complexity of model checking a **fixed FO formula** when the input is the model only.

Relationship between the complexity classes

The following relationships are known:

$$\mathrm{AC}^0 \subsetneq \textsc{LogSpace} \subseteq \textsc{NLogSpace} \subseteq \textsc{PTime} \subseteq \\ \subseteq \mathrm{NP} \subseteq \textsc{PSpace} \subseteq \\ \subseteq \textsc{ExpTime} \subseteq \textsc{NExpTime}$$

Moreover, we know that:

$$\textsc{PTime} \subsetneq \textsc{ExpTime}.$$

Complexity of First-Order Logic

Logical Formulae and Logical Theories

In mathematics, logic is used mostly to describe a set of “relevant” interpretations and prove theorems on these interpretations.

We define a **logical theory**, i.e., a finite set of logical formulae.

We look at interpretations satisfying the theory

- independently from the chosen assignment
- Interpretations satisfying a theory represent **valid possible “worlds”.**

We use the formal tools of equivalence and implication to prove our statements.

Implication and Equivalence (recap)

Let φ, ψ be two FOL formulae,

- φ **implies** ψ ($\varphi \models \psi$) if for every I, α s.t. $I, \alpha \models \varphi$ we have $I, \alpha \models \psi$
- φ **is equivalent to** ψ ($\varphi \equiv \psi$) if for every I, α, we have that $I, \alpha \models \varphi$ if and only if $I, \alpha \models \psi$

An Example of FOL Theory

1. $\forall x, y, z\ ((x + y) + z = x + (y + z))$, i.e., addition is associative.
2. $\forall x, y\ (x + y = y + x)$, i.e., addition is commutative.
3. $\forall x, y, z\ ((x \cdot y) \cdot z = x \cdot (y \cdot z))$, i.e., multiplication is associative.
4. $\forall x, y\ (x \cdot y = y \cdot x)$, i.e., multiplication is commutative.
5. $\forall x, y, z\ (x \cdot (y + z) = (x \cdot y) + (x \cdot z))$, i.e., multiplication distributes over addition.
6. $\forall x\ (x + 0 = x \wedge x \cdot 0 = 0)$, i.e., zero is an identity for addition, and an absorbing element for multiplication (actually superfluous[note 1]).
7. $\forall x\ (x \cdot 1 = x)$, i.e., one is an identity for multiplication.
8. $\forall x, y, z\ (x < y \wedge y < z \Rightarrow x < z)$, i.e., the '<' operator is transitive.
9. $\forall x\ (\neg(x < x))$, i.e., the '<' operator is irreflexive.
10. $\forall x, y\ (x < y \vee x = y \vee y < x)$, i.e., the ordering satisfies trichotomy.
11. $\forall x, y, z\ (x < y \Rightarrow x + z < y + z)$, i.e. the ordering is preserved under addition of the same element.
12. $\forall x, y, z\ (0 < z \wedge x < y \Rightarrow x \cdot z < y \cdot z)$, i.e. the ordering is preserved under multiplication by the same positive element.
13. $\forall x, y\ (x < y \Rightarrow \exists z\ (x + z = y))$, i.e. given any two distinct elements, the larger is the smaller plus another element.
14. $0 < 1 \wedge \forall x\ (x > 0 \Rightarrow x \geq 1)$, i.e. zero and one are distinct and there is no element between them.
15. $\forall x\ (x \geq 0)$, i.e. zero is the minimum element.

Logical Tasks: Complexity

- **Validity**: check whether a FOL formula is valid.
- **Satisfiability**: check whether a FOL formula is satisfiable.
- **Implication**: Check whether $\varphi \vDash \psi$, for input FOL formulae φ, ψ.

Logical Tasks: Complexity

- **Validity**: check whether a FOL formula is valid.
  - **Undecidable**

- **Satisfiability**: check whether a FOL formula is satisfiable.
  - **Undecidable**

- **Implication**: Check whether $\varphi \models \psi$, for input FOL formulae φ, ψ.
  - **Undecidable**

Unfortunately, in general none of the problems above is decidable.

Complexity of Propositional Logic

Truth and Falsity

Assume a propositional formula f.

- f is called **satisfiable** if $I(f)$ = **true,** for some propositional interpretation **I.**
- f is called **valid (tautology)** if $I(f)$ = **true,** for every propositional interpretation **I.**
- f is called **falsifiable** if $I(f)$ = **false,** for some propositional interpretation **I.**
- f is called **unsatisfiable** if $I(f)$ = **false,** for every propositional interpretation **I.**

Computational Complexity: Upper Bounds

- Checking whether a propositional formula f is **satisfiable** is in NP.
  - Guess an interpretation I (for the variables in f).
  - Check whether the I(f) = **true.**
- Checking whether a propositional formula f is **unsatisfiable** is in coNP.
  - Guess an interpretation I (for the variables in f).
  - Check whether the I(f) = **true.**
- Checking whether a propositional formula f is **falsifiable** is in NP.
  - Guess an interpretation I (for the variables in f).
  - Check whether the I(f) = **false.**
- Checking whether a propositional formula f is **valid** is in coNP.
  - Guess an interpretation I (for the variables in f).
  - Check whether the I(f) = **false.**

Normal Forms

- A propositional formula f is in n conjunctive normal form (n-CNF) if
  - $\mathbf{f = (l^1_1 \vee l^1_2 \vee \ldots \vee l^1_n) \wedge \ldots \wedge (l^m_1 \vee l^m_2 \vee \ldots \vee l^m_n)}$
  - Where each l^i_j is either an atom or its negation.
  - Every propositional formula has an equivalent formula in 3-CNF
  - However, the equivalent may be exponentially larger.

- A propositional formula f is in n disjunctive normal form (n-DNF) if
  - $\mathbf{f = (l^1_1 \wedge l^1_2 \wedge \ldots \wedge l^1_n) \vee \ldots \vee (l^m_1 \wedge l^m_2 \wedge \ldots \wedge l^m_n)}$
  - Where each l^i_j is either an atom or its negation.
  - Every propositional formula has an equivalent formula in 3-DNF
  - However, the equivalent may be exponentially larger.

Computational Complexity: Lower Bounds

- Checking whether a propositional formula is **satisfiable** is NP-Hard.
  - Even for formulae in 3-CNF
- Checking whether a propositional formula is **unsatisfiable** is coNP-Hard.
  - Even for formulae in 3-DNF
- Checking whether a propositional formula is **falsifiable** is NP-hard.
  - Even for formulae in 3-DNF.
- Checking whether a propositional formula is **valid** is in coNP-Hard.
  - Even for formulae in 3-CNF

Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.