# Set Theory
# &
# Propositional Logic

David Pereira

CISTER Research Centre / DEI - ISEP , P.Porto

October 27, 2021

# What is in the menu for today?

Set Theory

Propositional Logic

# Table of Contents

# What is a set?

## Definition

Sets are collections of elements, elements of which can be of any kind of mathematical object, e.g., integers, symbols, variables, other sets, etc.

## Example

Some examples of sets that relate to requirements specification:

- all the oceans on earth
- the passwords that may be generated by a certain cryptographic algorithm
- the log files produced by a set of monitors
- the virtual machines and containers to be deployed in a certain cloud environment
- . . .

# Defining sets and set membership

## If the set is small...

... we can define it ***by extension***, that is, we ***enumerate all its elements***. We use braces '{' and '}' to delimit the set, and use ',' to separate its elements. For instance, we can represent all the oceans on earth as follows:

$$Oceans = \{Atlantic, Pacific, Indian, Arctic\}$$

## Finiteness and cardinality

A set can be finite, if the number of its elements is also finite; otherwise the set is infinite. When the set is finite, its size, or ***cardinality*** is denoted by $|S|$.

## Set membership

If $S$ is a set, and $x$ an element whose type is the same as the elements of $S$, then we denote by $x \in S$ the relation of $x$ ***being a member*** of $S$. We can also state that "$x$ ***is in*** $S$"

$$Atlantic \in Oceans,\ Mediterranean \notin Oceans$$

# Set equality

## Extensionality

Two sets $S$ and $T$ are equal, which we denote by $S = T$, if and only if **they have the same elements**, that is:

1. if for all $x$ we have $x \in S$ then $x \in T$, and
2. if for all $y$ we have $y \in T$ then $y \in S$.

We can establish a rule helps synthesise the above condition, and that guides reasoning:

$$\frac{(\forall x, x \in S \rightarrow x \in T) \wedge (\forall x, x \in T \rightarrow x \in S)}{S = T}$$

## Another view on set membership (extensionality of membership)

If $x = e_1$ or $x = e_2$ or $\ldots$ or $x = e_n$, then it holds that $x \in \{e_1, e_2, \ldots, e_n\}$.

# The empty set and other set relations

## Definition

A set $S$ is said to be *empty*, denoted $S = \emptyset$, if it **has no elements**.
If $S = \emptyset$, then for all elements $x$, $x \notin S$.

## Subset

If we know that **all members** $x$ of a set $S$ **are also members of a set** $T$, then we say that $S$ is a *subset* of $T$, which we denote this relation by $S \subseteq T$.

## Subset and set equality

Let $S$ and $T$ be two sets such that $S \subseteq T$ and $T \subseteq S$ holds. Then, $S = T$ holds.

# Set Comprehension

## Definition

If $S$ is a set and $\mathcal{P}$ is a property, then we can build a new set $T$ from $S$ by selecting only those for which $\mathcal{P}$ holds. We say that in this case we define $T$ by set comprehension and denote it by

$$\{x \in S \mid \mathcal{P}\}.$$

We can informally state that the left part $x \in S$ represents a "generator" whereas the right part ($\mathcal{P}$) represents a filter.

## Extending set comprehension

With the notion of set comprehension we can generate a new set from a given set $S$, a property $\mathcal{P}$ and a function that produces a new value from the elements of $S$ that satisfy $\mathcal{P}$. Let $f$ be such a function. Then we denote this extended notion of comprehension by

$$\{x \in S \mid \mathcal{P} \bullet f\}.$$

# Example

Let $S$ represent the set of all ISEP's students. Let *MESCC* be the property of being enrolled in MESCC and *RAMDE* be the property of being enrolled in RAMDE. Hence, we can define the set of all of the MESCC's students that are attending RAMDE's classes:

$$S_{RAMDE} = \{s \in S \mid MESCC(s) \text{ and } RAMDE(s)\}$$

We might be interested in getting some statistics, e.g., gender balance. Assuming a function *gender* that maps names into either *male* or *female*, we can define

$$GB_{RAMDE} = \{s \in S \mid MESCC(s) \text{ and } RAMDE(s) \bullet gender\},$$

or simply by

$$GB_{RAMDE} = \{s \in S_{RAMDE} \mid True \bullet gender\}$$

# Lets look at some examples

## Example

Let $S$ be the set of natural numbers smaller or equal to 10. We can represent this as a set comprehension expression as follows

$$S = \{x \in \mathbb{N} \mid x \leq 10\}$$

. We can also write this easily in Python

```
>>> S = set(range(11))
>>> S
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

# Lets look at some more simple examples

## Example

Lets use $S$ to build a new subset of it, the set of even natural numbers smaller or equal than 10. We express this set by comprehension as follows

$$Even_{10} = \{x \in S \mid x \, mod \, 2 \equiv 0\},$$

or in Python:

```
>>> even10 = [x for x in S if x%2 == 0]
>>> even10
{2, 4, 6, 8, 10}
```

# Lets look at some more simple examples

## Example

Now we have the set $enve_{10}$ that contains all the even natural numbers that are lower or equal to 10. Using the extended set comprehension notation, we can build a new set that, for instance, has the doubles of the values of $even_{10}$. We define it simply by assuming a function $f(x) = x \times 2$ and defining

$$Double\_Even_{10} = \{x \in S \mid x \bmod 2 \equiv 0 \bullet f\},$$

or in Python:

```
>>> even10 = [x*2 for x in S if x%2 == 0]
>>> even10
{4, 8, 12, 16, 20}
```

# Set Operations - intersection and union

## Intersection

The intersection of two sets $S$ and $T$ is denoted by $S \cap T$ and is the set with the elements that are common to both sets. If the sets are disjoint, then $S \cap T = \emptyset$.

$$\frac{x \in (S \cap T)}{x \in S \wedge x \in T}$$

## Union

The union of two sets $S$ and $T$ is denoted by $S \cup T$ and is the set containing all the elements of $S$ and all the elements of $T$.

$$\frac{x \in (S \cup T)}{x \in S \vee x \in T}$$

# Examples

Example (Intersection)

Example (Union)

# Set Operations - difference

## Difference

The difference between two sets is denoted by $S \setminus T$ and refers to the set with all the elements of $S$ that are not members of $T$.

$$\frac{x \in (S \setminus T)}{x \in S \wedge x \notin T}$$

## Example (Small exercise)

Let $S = \{1, 2, \ldots, 10\}$ and let $T = \{0, 1, 2, \ldots, 10\}$. Then, depending on the way we calculate the difference between these sets, we get different results:

$$T \setminus S = \{0\} \text{ but } S \setminus T = \emptyset$$

# Powerset

## Definition

The powerset of a set $S$, denoted $\mathbb{P}(S)$, is the set containing all the subsets of $S$.

## Example (Simple exercise)

Let $S = \{0, 1\}$. Then $\mathbb{P}(S) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$

## A note on finite sets and cardinality

If $S$ is a finite set, then we denote its cardinality by $|S|$. In the case of the powerset, if $|S| = n$, then $|\mathbb{P}(S)| = 2^n$.

## A logical rule for the powerset

$$\frac{S \subseteq T}{S \in \mathbb{P}(T)}$$

# Cartesian Product

## Definition

If $S$ and $T$ are sets, then their cartesian product $S \times T$ is formed by the set of all pairs $(x, y)$ such that $x \in S$ and $y \in T$.

## Tuples and $n$-tuples

We can extend the cartesian product to more than two sets. That is, consider sets $S_1, \ldots, S_n$ where $n > 2$. The elements of this product $S_1 \times \cdots S_n$ are called $n$-tuples and have the form $(x_1, \ldots, x_n)$ such that $x_1 \in S_1$, $x_2 \in S_2$, and so forth.

## A rule for reasoning about cartesian products...

$$\frac{x_1 \in S_1 \wedge \cdots \wedge x_n \in S_n}{(x_1, \ldots, x_n) \in S_1 \times \cdots \times S_n}$$

# Exercising a little bit

Lets recall the first examples of sets mentioned during the class:

1. *all the oceans on earth* $\Rightarrow$ we already solved this one!
2. *the passwords that may be generated by a certain cryptographic algorithm*
3. *the log files produced by a set of monitors*
4. *the virtual machines and containers to be deployed in a certain cloud environment*

## Exercise

Formalize each of the examples presented above.

# Exercising a little bit

*the passwords that may be generated by a certain cryptographic algorithm*

This requirement is far too abstract. Since there is no more information of the domain, we assume a component that takes as inputs identifiers of users from a set $I$, and returns passwords for from a set $P$, using a cryptographic algorithm $A$.

$$Passwd \;=\; \{p \in P \mid \exists\, id \in I,\; A(id) = p\}$$

or, alternatively,

$$Passwd \;=\; \{id \in I \mid True \bullet A(id)\}$$

# Exercising a little bit

*the log files produced by a set of monitors*

To formalize this requirement, we need a way to have access to monitors in order to obtain the list of associated log files. If $M$ is the set of monitors, and a log file is denoted by its name, i.e., a string, we assume such a function $logs : M \rightarrow String$.
The set of all log files produced by a set of monitors, in this case $M$, is

$$Logs = \{m \in M \bullet logs(m)\}$$

# Exercising a little bit

*the containers to be deployed in a certain cloud environment*

We assume two properties on the containers: (i) that a container can be check for sucessful deployment in a given clud platform; (ii) that it is planned for deployment already. So, lets assume the properties:

- $isdep(c, i)$ meaning that container $c$ is deployable on cloud infrastructure $i$,
- $planned(c)$ meaning that the deployment of $c$ is planned to occur.

Then, we can define our set as follows:

$$DC = \{c \in C \mid \exists i,\ isdep(c, i) \text{ and } planned(c)\}$$

# Table of Contents

# Propositional Logic - Syntax

## Syntax

Let $\mathcal{B} = \{b_0, b_1, \ldots\}$ be a set of (truth/Boolean) varibles and $\mathcal{C} = \{\neg, \wedge\}$ be a set of logical connectives. The language of propositional logic, denoted $\mathcal{L}_P$, is recursively/inductively defined by:

$$\varphi, \varphi_1, \varphi_2 \ ::= \ \top \mid b \in \mathcal{B} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2$$

The remaining logical operators are defined as follows:

- (False) $\bot \equiv \neg\top$
- (Disjunction) $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$
- (Implication) $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
- (Equivalence) $\varphi_1 \leftrightarrow (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$

# Propositional Logic - Semantics

## Truth valuations

Let $\mathcal{B} = \{b, b_0, b_1, \ldots\}$ be a set of variables and let $\mathbb{B} = \{\texttt{tt}, \texttt{ff}\}$ the set of truth values/constants. A valuation function is a function $v : \mathcal{B} \to \mathbb{B}$ that associates values in $\mathbb{B}$ to variables in $\mathcal{B}$.

## Formula evaluation

Considering the truth valuation $v : \mathcal{B} \to \mathbb{B}$, the evaluation of the truth value of a formula in $\mathcal{L}_{LP}$ is given by the recursive function $f_{LP} : \mathcal{L}_{LP} \to \mathbb{B}$, defined as follows:

- $f_{LP}(\top) = \texttt{tt}$
- $f_{LP}(b) = v(b)$
- $f_{LP}(\neg\varphi) = \neg f_{LP}(\varphi)$
- $f_{LP}(\varphi_1 \wedge \varphi_2) = f_{LP}(\varphi) \wedge f_{LP}(\varphi_2)$

# Propositional Logic - Semantics

## Satisfiability

Let $\varphi$ be a $\mathcal{L}_{LP}$ formula. We say that $\varphi$ is **satisfiable** if, and only if there exists a valuation $v : \mathcal{B} \to \mathbb{B}$ for variables in $\varphi$ such that $f_{LP}(\varphi) = \texttt{tt}$.

## Example

Let $b_0, b_1 \in \mathcal{B}$ and let $\varphi = b_0 \vee b_1$. Whenever we have $f(b_0) = \texttt{tt}$, or $f(b_1) = \texttt{tt}$, or both, then we also have $f_{LP}(\varphi) = \texttt{tt}$

| $b_0$ | $b_1$ | $f_{LP}(\varphi)$ |
|---|---|---|
| ff | ff | ff |
| ff | tt | tt |
| tt | ff | tt |
| tt | tt | tt |

# Propositional Logic - Semantics

## Validity

A formula $\varphi$ is said to be valid if, and only if $f_{LP}(\varphi) = \texttt{tt}$ for all valuation function $\nu : \mathcal{B} \to \mathbb{B}$.

## Example

Let $b \in \mathcal{B}$ and let $\varphi = b \vee \neg b$. Whatever the attribution of truth values to $b$ by a valuation function $f$, we always get $f_{LP}(\varphi) = \texttt{tt}$

| $b_0$ | $b_1$ | $f_{LP}(\varphi)$ |
|---|---|---|
| ff | ff | tt |
| ff | tt | tt |
| tt | ff | tt |
| tt | tt | tt |

Note: recall that $\varphi \vee \psi$ holds if at least one of the formulae holds.

# Propositional Logic - Semantics

## Contradiction

A formula $\varphi \in \mathcal{L}_{LP}$ is called a contradiction if, and only if $f_{LP}(\varphi) = \texttt{ff}$ for all possible valuation function $v : \mathcal{B} \to \mathbb{B}$.

## Example

Let $b \in \mathcal{B}$ and let $\varphi = b \wedge \neg b$. Whatever the attribution of truth values to $b$ by a valuation function $f$, we always get $f_{LP}(\varphi) = \texttt{ff}$

| $b_0$ | $b_1$ | $f_{LP}(\varphi)$ |
|---|---|---|
| ff | ff | ff |
| ff | tt | ff |
| tt | ff | ff |
| tt | tt | ff |

Note: recall that $\varphi \wedge \psi$ holds if both formulas hold.

# Propositional Logic - Semantics

Important relation between satisfiability, validity, and contradiction

- we can verify if $\varphi$ is valid if we are able to conclude that $\neg\varphi$ is a contradiction.

# Propositional Logic - Normal Forms

## Literal

A **literal** is a variable $b \in \mathcal{B}$ or its negation, i.e., $\neg b$.

## Negative Normal Form (NNF)

These are LP formulae where the connective "$\neg$" is applied only onto literals.

## Example

For instance, the formula

$$(\neg b_1 \wedge \neg b_2) \vee b_1$$

is in NNF. However, the formula

$$\neg((b_1 \vee b_2) \wedge b_1)$$

is not, although they are equivalent formulae (lets check that?).

# Propositional Logic - Normal Forms

## Disjunctive Normal Form (DNF)

These are the formulas of LP que satisfy the following syntactic pattern:

$$(\varphi_{11} \wedge \ldots \wedge \varphi_{1k_1}) \vee \ldots \vee (\varphi_{n1} \wedge \ldots \wedge \varphi_{nk_n})$$

where each $\varphi_{ij}$ is a literal.

## DNF and satisfiability

A DNF formula is satisfiable if, and only if one of its inner conjunctions is satisfiable.

# Propositional Logic - Normal Forms

## Conversion to DNF

In general, the following approach allows us to pick up a formula of LP and transform it into a DNF form: (i) produce and equivalent form containing only the connectives $\wedge$, $\vee$, and $\neg$; (ii) transform the formula into a NNF; and (iii) apply distributivity:

$$\begin{aligned}(\varphi \vee \psi) \wedge \theta &= (\varphi \wedge \theta) \vee (\psi \wedge \theta) \\ \varphi \wedge (\psi \vee \theta) &= (\varphi \wedge \psi) \vee (\varphi \wedge \theta)\end{aligned}$$

| $\varphi$ | $\psi$ | $\theta$ | $(\varphi \vee \psi)$ | $(\varphi \vee \psi) \wedge \theta$ | $(\varphi \wedge \theta)$ | $(\psi \wedge \theta)$ | $(\varphi \wedge \theta) \vee (\psi \wedge \theta)$ |
|---|---|---|---|---|---|---|---|
| ff | ff | ff | ff | ff | ff | ff | ff |
| ff | ff | tt | ff | ff | ff | ff | ff |
| ff | tt | ff | tt | ff | ff | ff | ff |
| ff | tt | tt | tt | tt | ff | tt | tt |
| tt | tt | tt | tt | tt | tt | tt | tt |
| tt | ff | ff | tt | ff | ff | ff | ff |
| tt | ff | tt | tt | tt | ff | tt | tt |
| tt | tt | ff | tt | ff | ff | ff | ff |

# Propositional Logic - Normal Forms

## Conjunctive Normal Form (CNF)

These are LP formulae that satisfy the following syntactic pattern:

$$(\varphi_{11} \vee \ldots \varphi_{1k_1}) \wedge \ldots \wedge (\varphi_{n1} \vee \ldots \varphi_{nk_n})$$

where each $\varphi_{ij}$ is a literal.

## CNF e validity

A disjunction of literais

$$\varphi_{11} \vee \ldots \varphi_{1n}$$

is valid if, and only if $l_i$ e $\neg l_i$ are part of the same disjunction of literals. To verify if a CNF formula is valid, it is enough to check all its disjunctions are themselves valid.

# Propositional Logic - Normal Forms

## Conversion to CNF

In general, we should follow a similar approach as the one for DNF, but using distribution equivalences for conjunction, that is:

$$\begin{aligned}(\varphi \wedge \psi) \vee \theta &= (\varphi \vee \theta) \wedge (\psi \vee \theta) \\ \varphi \vee (\psi \wedge \theta) &= (\varphi \vee \psi) \wedge (\varphi \vee \theta)\end{aligned}$$

## CNF and satisfiability - worst case

The satisfiability problem can be solved via the generation of the corresponding truth table. In the worst case, if the formula has *n* propositional variables, then the algorithmic complexity will be $2^n$.

## CNF and satisfiability - alternatives

In general, there are alternative algorithms that mitigate this exponential growth. Such algorithms are particularly efficient if a formula is in CNF form.

We will dive into one of those algorithms in the next class... the David-Puttman algorithm.

# Natural Deduction

We will now start looking at "strange things" like the one presented below...

| | | |
|---|---|---|
| 1 | $A$ | |
| 2 | $\quad B$ | assumption |
| 3 | $\quad A$ | repetition (1) |
| 4 | $B \rightarrow A$ | $\rightarrow$-introduction (2-3) |
| 5 | $A \rightarrow (B \rightarrow A)$ | $\rightarrow$-introduction (1-4) |

This, and similar representations are called "deductions" or "truth derivations", or "proofs" (in this case, written in Fitch notation).

# Natural Deduction

## What is natural deduction

It is a finite sequence of steps, where each step amounts at the application of a deduction rule over formulas that are already in the "context". Some of these hypothesis can be inserted as assumptions in the proof context, although they are eliminated at some point during the proof construction process.

## Introduction and elimination rules

Each connective has associated both introduction and elimination rules:

- **introduction rules:** introduce a new formula, through a construction oriented to a certain connective, and based on formulae already in the proof context (or making an assumption)
- **elimination rules:** eliminate a formula and generates either new or sub-formulas from existing ones, based on the target connective of the formula that we are eliminating

# Natural Deduction Rules - Conjunction

## $\wedge$ introduction

We can construct a new conjunction $\varphi \wedge \psi$ if we know that both $\varphi$ and $\psi$ are valid in the proof context

$$\frac{\varphi \qquad \psi}{\varphi \wedge \psi} \wedge\text{-intro}$$

## Elimination rules

If we know already that $\varphi \wedge \psi$ holds, then we can "eliminate" one of formulas.

$$\frac{\varphi \wedge \psi}{\varphi} \wedge\text{-elimL}$$

$$\frac{\varphi \wedge \psi}{\psi} \wedge\text{-elimR}$$

# Natural Deduction Rules - Disjunction

## Introduction of disjunction

We can construct a new conjunction $\varphi \vee \psi$ if we know that either $\varphi$ or $\psi$ hold.

$$\frac{\varphi}{\varphi \vee \psi} \vee\text{-introL} \qquad \frac{\psi}{\varphi \vee \psi} \vee\text{-introR}$$

## Elimination rules

The elimination, in this case, assumes the form of introducing a new formula $\theta$ in case we can derive $\theta$ from both $\varphi$ and $\psi$, and we know that $\varphi \vee \psi$ holds.

$$\frac{\varphi \vee \psi \qquad \begin{array}{c}[\varphi] \\ \vdots \\ \theta\end{array} \qquad \begin{array}{c}[\psi] \\ \vdots \\ \theta\end{array}}{\theta} \vee\text{-elim}$$

# Natural Deduction Rules - Negation

## Introduction of negation

If from $\varphi$ we can derive false, then we can conclude that $\varphi$ does not hold.

$$\dfrac{\begin{array}{c}[\varphi] \\ \vdots \\ \mathtt{ff}\end{array}}{\neg\varphi}\ \neg\text{-intro}$$

## Elimination of negation

From false, we can conclude whatever we want.

$$\dfrac{\mathtt{ff}}{\varphi}\ \neg\text{-elim}$$

# Natural Deduction Rules - Implication

## Introduction of implication

If we assume $\varphi$ and we can derive $\psi$ from it, then we can conclude that $\varphi \to \psi$.

$$\dfrac{\begin{array}{c}[\varphi] \\ \vdots \\ \psi\end{array}}{\varphi \to \psi} \to\text{-intro}$$

## Elimination of implementation

From false, we can conclude whatever we want.

$$\dfrac{\varphi \to \psi \qquad \varphi}{\psi} \to\text{-elim}$$

# Natural Deduction Rules - Structural Rule

## The rule of repetition

This is a structural rule, it does not allow to conclude anything new besides repeating a formula that exists already in the context.

$$\frac{\varphi}{\varphi}\ R$$

# Some derived rules

$$\frac{\varphi \to \psi \qquad \neg\psi}{\neg\varphi}\ \text{MT}$$

$$\frac{\varphi}{\neg\neg\varphi}\ \neg\neg\text{-intro}$$

$$\frac{\begin{array}{c}[\neg\varphi]\\ \vdots \\ \mathtt{ff}\end{array}}{\varphi}\ \text{RA}$$

$$\frac{}{\varphi \vee \neg\varphi}\ \text{ET}$$

# The End