# Computational Logic - Assignment 1

Alessandro Marostica

October 13, 2023

## Prelude

I have never used LaTeX before, please bear with my plausibly horrible usage.

## Exercise 2

**Disjunction** Using De Morgan's law, one can define the OR operator as such:

$$A \vee B \dashv\vdash \neg(\neg A \wedge \neg B)$$

| $A$ | $B$ | $A \vee B$ |
|---|---|---|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

| $A$ | $B$ | $\neg A$ | $\neg B$ | $\neg A \wedge \neg B$ | $\neg(\neg A \wedge \neg B)$ |
|---|---|---|---|---|---|
| T | T | F | F | F | T |
| T | F | F | T | F | T |
| F | T | T | F | F | T |
| F | F | T | T | T | F |

**Implication** One can define IMPLIES as such:

$$A \to B \dashv\vdash \neg A \vee B$$

| $A$ | $B$ | $A \to B$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

| $A$ | $B$ | $\neg A$ | $\neg A \vee B$ |
|---|---|---|---|
| T | T | F | T |
| T | F | F | F |
| F | T | T | T |
| F | F | T | T |

**Material equivalence** One can define IFF as such:

$$A \dashv\vdash B \dashv\vdash (A \wedge B) \vee (\neg A \wedge \neg B)$$

| $A$ | $B$ | $A \dashv\vdash B$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | T |

| $A$ | $B$ | $A \wedge B$ | $\neg A$ | $\neg B$ | $\neg A \wedge \neg B$ | $(A \wedge B) \vee (\neg A \wedge \neg B)$ |
|---|---|---|---|---|---|---|
| T | T | T | F | F | F | T |
| T | F | F | F | T | F | F |
| F | T | F | T | F | F | F |
| F | F | F | T | T | T | T |

Note that IFF yields the same result as XNOR, which is opposite of XOR.

**Exclusive disjunction** One can define XOR as such:

$$A \oplus B \dashv\vdash (A \wedge \neg B) \vee (\neg A \wedge B)$$

| $A$ | $B$ | $A \oplus B$ |
|---|---|---|
| T | T | F |
| T | F | T |
| F | T | T |
| F | F | F |

| $A$ | $B$ | $\neg A$ | $\neg B$ | $A \wedge \neg B$ | $\neg A \wedge B$ | $(A \wedge \neg B) \vee (\neg A \wedge B)$ |
|---|---|---|---|---|---|---|
| T | T | F | F | F | F | F |
| T | F | F | T | T | F | T |
| F | T | T | F | F | T | T |
| F | F | T | T | F | F | F |

**Defining simple operators with NAND**
NAND is defined as such:

$$p \uparrow q \dashv\vdash \neg(p \wedge q)$$

hence:

$\boldsymbol{\neg p}$ can be defined as $p \uparrow p$ ($\uparrow$ is the symbol for NAND), this is equivalent to $p \uparrow p \dashv\vdash \neg(p \wedge p) \dashv\vdash \neg p$. In natural language this can be referred to as "not both p and p".

$\boldsymbol{p \wedge q}$ can be defined as $\neg(p \uparrow q)$, this is equivalent to $\neg(p \uparrow q) \dashv\vdash \neg(\neg(p \wedge q)) \dashv\vdash p \wedge q$

$\boldsymbol{p \vee q}$ can be defined as $(p \uparrow p) \uparrow (q \uparrow q)$, this is equivalent to $(p \uparrow p) \uparrow (q \uparrow q) \dashv\vdash \neg(p \wedge p) \uparrow \neg(q \wedge q) \dashv\vdash \neg p \uparrow \neg q \dashv\vdash \neg(\neg p \wedge \neg q) \dashv\vdash p \vee q$. In the last step De Morgan's law is applied.

$\boldsymbol{p \rightarrow q}$ can be defined as $p \uparrow (q \uparrow q)$, this is equivalent to $p \uparrow (q \uparrow q) \dashv\vdash p \uparrow \neg q \dashv\vdash \neg p \vee \neg\neg q \dashv\vdash \neg p \vee q \dashv\vdash p \rightarrow q$. In this proof the rules of disjunction in terms of NAND and negation in terms of NAND are applied.

$\boldsymbol{p \leftrightarrow q}$ can be defined as $((p \uparrow p) \uparrow (q \uparrow q)) \uparrow (p \uparrow q)$, this is equivalent to $((p \uparrow p) \uparrow (q \uparrow q)) \uparrow (p \uparrow q) \dashv\vdash \neg(((p \uparrow p) \uparrow (q \uparrow q)) \wedge (p \uparrow q)) \dashv\vdash \neg((p \vee q) \wedge (p \uparrow q)) \dashv\vdash \neg((p \vee q) \wedge \neg(p \wedge q)) \dashv\vdash \neg(p \oplus q) \dashv\vdash p \leftrightarrow q$. In this proof many rules are applied: disjunction interms of NAND, definition of XOR, XOR is negation of biconditional.

## Exercise 3

$\boldsymbol{p \wedge (q \wedge r) \vdash (p \wedge q) \wedge r}$ This sequent can be proved in this way:

| | | |
|---|---|---|
| 1 | $p \wedge (q \wedge r)$ | Premise |
| 2 | $q \wedge r$ | $\wedge e1$ |
| 3 | $q$ | $\wedge e2$ |
| 4 | $r$ | $\wedge e2$ |
| 5 | $p \wedge q$ | $\wedge i1, 3$ |
| 6 | $(p \wedge q) \wedge r$ | $\wedge i5, 4$ |

$\neg p \land \neg q \vdash \neg(p \lor q)$ This sequent can be proved by contradiction in this way:

| | | |
|---|---|---|
| 1 | $\neg p \land \neg q$ | Premise |
| 2 | $p \lor q$ | Assumption for the sake of contradiction |
| 3 | $\neg p$ | $\land e1$ |
| 4 | $\neg q$ | $\land e1$ |
| 5 | $p$ | $\lor e2$ |
| 6 | $\bot$ | 5, 3 |
| 7 | $\neg(p \lor q)$ | Proof by contradiction |

## Exercise 4

**Intuitionistic Logic** Intuitionistic logic can be, in a way, understood as a weakening of classical logic: it gives the reasoner less possibilities to infer informations based on certain propositions and it does not allow any new inference that could not be made under classical logic. Intuitionistic logic differs from classical propositional logic in its treatment of logical connectives, namely:

- **Negation:** in this kind of logic negation is defined as "A is not provable" rather than "A is false", this is because, in intuitionistic logic, a proposition can be true, false or unknown. This implies that, in intuitionistic logic, the Law of the Excluded Middle, which states that every proposition is either $A$ or $\neg A$ does not hold.
- **Implication:** in intuitionistic logic, implication is interpreted ad "if we can prove A, then we can prove B", an example of constructive interpretation. In contrast, classical logic interprets implication as the preservation of truth.
- **Double negation:** because of the different interpretation of negation, double negation does not hold in intuitionistic logic, one cannot always infer a propositon from its double negation

In intuitionistic propositional logic, propositonal formulas are only considered true when there is direct evidence, hence proof. Operations therefore preserve justification rather than truth. Intuitionistic logic found applications in certain computerized tools, namely proof assistants, which aid users in the generation of large scale proofs, enabling mathematicians and logicians to develop and prove extremely complex systems, beyond human capability. It can also be used to formally verify correctness of computer programs, especially in functional programming, a paradigm which encourages a constructive approach.

## Exercise 3

**Disjunctive Normal Form** is a way of expressing propositional formulas as the logical OR of multiple clauses which are conjunctions of one or more literals. As in CNF, the only logical operators used are $\land$, $\lor$ and $\neg$. Some examples of formulas in DNF follow:

- $A$
- $(A \land B)$
- $(A \land B) \lor C$
- $(A \land B) \lor (C \land D)$

- $(A \wedge \neg B \wedge \neg C) \vee (D \wedge E)$

Also as in CNF, all formulas can be converted into equivalent DNF formulas but in some cases it can lead to an exponential growth of the formula. A DNF formula is satisfiable if, and only if, one of its conjuncts is satisfiable, a problem solvable in polynomial time. This is contrasting to the satisfiability problem for CNF, which is an NP-hard problem. As for CNF, DNF simplifies complex proposotional formulas, which can then be evaluated in linear time. This also implies its usefulness in model checking, although after the conversion of such model in DNF. The following pseudo-code computes a generic propositional formula into DNF[1]:

```
fn DNF(formula):
    if formula == A:
        return formula
    else if formula == A ∧ B:
        return DNF(A) ∧ DNF(B)
    else if formula == A ∨ B:
        return DNF(A) ∨ DNF(B)
    else if formula == A → B:
        return DNF(¬A) ∨ DNF(B)
    else if formula == ¬A:
        return DNF(A)
```

[1] I am not sure about this pseudo-code, I couldn't find much information