University of Central Lancashire
UCLan

CO2412 Computational Thinking

Logical operators
Truth tables

Where opportunity creates success

What do you associate with "Logic"?

Logical operators

CO2412

25th January 2022

Boolean expressions in programming

In Python: False/True, but also 0/1, etc.

Similar in C/C++ and Java (false, true, 0, 1, …)

Python: and, or, not.

Languages closer to C: &&, ||, !

Boolean variables

p = True
$q = (i < j)$
r = False

Control flow

if q:
  …
else:
  …

while (p **and** q **and not** r):
  …

Propositional logic

Propositional logic is a standard way of denoting Boolean expressions.

The equivalent of a Boolean variable in propositional logic is an **atomic statement**.

Logical expressions that combine statements (say, statements R, S) into one (say, "R and S," denoted $R \wedge S$) are composite statements.

The same **Boolean operators** are used as in programming, but using a different notation:

$\neg$	$\wedge$	$\vee$
not	and	or

Boolean variables

p = True
$q = (i < j)$
r = False

Control flow

if q:
  …
else:
  …

while (p **and** q **and not** r):
  …

Propositional logic

Negation (¬)

Logical not is a unary operator, it has one argument. The negation of a statement S is denoted $\neg S$.

$\neg S$ is True if S is False, and vice versa.

$\neg p \lor q$ means "not-p or q"

$\neg(p \lor q)$, meaning "not (p or q)", requires parentheses

$(p \lor q) \lor r$ means the same as $p \lor (q \lor r)$

we can then just write $p \lor q \lor r$

Disjunction (∨)

Logical or is a binary operator, it has two arguments. The disjunction of two statements R, S is denoted $R \lor S$.

$R \lor S$ is False if both R and S are False, otherwise $R \lor S$ is True.

Precedence of operators:

Unary (negation) first, binary (all the others) last. You must use parentheses to indicate the precedence of multiple binary logical operators.

Propositional logic

Negation (¬)

Logical not is a unary operator, it has one argument. The negation of a statement S is denoted $\neg S$.

$\neg S$ is True if S is False, and vice versa.

Disjunction (∨)

Logical or is a binary operator, it has two arguments. The disjunction of two statements R, S is denoted $R \vee S$.

$R \vee S$ is False if both R and S are False, otherwise $R \vee S$ is True.

Conjunction (∧)

Logical and is a binary operator. The conjunction of two statements R, S is denoted $R \wedge S$.

$R \wedge S$ is True if both R and S are True, otherwise $R \wedge S$ is False.

Implication (→)

Logical implication (or **conditionality**) is a binary operator. "R implies S" is denoted $R \rightarrow S$.

$R \rightarrow S$ is False if R is True and S is False, otherwise it is True.

Propositional logic

Equivalence ($\leftrightarrow$)

Logical equivalence (or **biconditionality**) is a binary operator. "R equivalent S" is denoted $R \leftrightarrow S$.

$R \leftrightarrow S$ is True if R and S have the same truth value, otherwise it is False.

$R \leftrightarrow S$

can be understood as an abbreviation for

$(R \rightarrow S) \land (S \rightarrow R)$

or an abbreviation for

$(R \land S) \lor (\neg R \land \neg S)$

$R \rightarrow S$

can be understood as an abbreviation for

$\neg R \lor S$

or an abbreviation for

$\neg(R \land \neg S)$

Implication ($\rightarrow$)

Logical implication (or **conditionality**) is a binary operator. "R implies S" is denoted $R \rightarrow S$.

$R \rightarrow S$ is False if R is True and S is False, otherwise it is True.

Truth tables

CO2412

25th January 2022

I want this to make sense! Where are the examples?

We aim at being able to express statements from systems specifications, *e.g.*,

"The item is not handed to the customer unless a payment has been received."

Propositional logic is rather weak, we will have to enrich it for this purpose and look at stronger logics. This gives us only the very form of such statements.

But we might begin with:

- p defined by "the item is handed to the customer."
- q defined by "the payment has been received."

in this form, the actual content has disappeared … **:(**

Then our statement might be expressed in propositional logic by $p \to q$.

Using predicates, which we will look into later, this might become

$$\text{HandoverItem}(\text{self}, \text{customer}) \to \text{PaymentDone}(\text{customer}, \text{self}).$$

Semantics of propositional logic

Three branches of the theory of formal languages:

- **Syntax** (theory of the **structure** of language)
- **Semantics** (theory of the **meaning** of language)
- **Pragmatics** (theory of the **use** of language)

Generally speaking, semantics refers to "meaning," as opposed to syntax, which refers to "proper grammar and notation."

Under many typical circumstances (particularly in computing), a code, formula, statement, *etc.*, can only have a semantic content if it has correct syntax. However, human language pragmatics permits people to also make sense of utterances that are not grammatically correct.

Just like statements in human language, logical statements use language(s). They can be analysed in the same way, and so can programming languages.

Semantics of propositional logic

The semantics (meaning) of a propositional logic statement is given by the **valuations**, truth value assignments for atomic statements, that make it true.

The straightforward way of expressing that is through a **truth table**:

p	q	"p implies q" $p \to q$
False	False	True
False	True	True
True	False	False
True	True	True

Task: Determine truth table for $(p \to q) \wedge (q \vee r)$

recall that $p \to q$ *"p implies q"*
can be rewritten as

$\neg p \vee q$ *"not-p or q"*

or also as

$\neg(p \wedge \neg q)$ *"not (p and not-q)"*

Semantics of propositional logic

The semantics (meaning) of a propositional logic statement is given by the **valuations**, truth value assignments for atomic statements, that make it true.

The straightforward way of expressing that is through a **truth table**:

			"p implies q"	*"q or r"*	*"(p implies q) and (q or r)"*
p	q	r	$p \rightarrow q$	$q \vee r$	$(p \rightarrow q) \wedge (q \vee r)$
False	False	False	True	False	False
False	False	True	True	True	True
False	True	False	True	True	True
False	True	True	True	True	True
True	False	False	False	False	False
True	False	True	False	True	False
True	True	False	True	True	True
True	True	True	True	True	True

Satisfiability

Typically, we would expect the truth value of a propositional logic statement to depend on the valuation, *i.e.*, on the truth values of its atomic statements.

But there are **statements that can never become true**. They are **unsatisfiable**.

p	q	r	*"not (p implies q)"* $\neg(p \rightarrow q)$	$\neg(q \rightarrow r)$ *"not (q implies r)"*	*"not (p implies q) and not (q implies r)"* $\neg(p \rightarrow q) \wedge \neg(q \rightarrow r)$
False	False	False	False	False	False
False	False	True	False	False	False
False	True	False	False	True	False
False	True	True	False	False	False
True	False	False	True	False	False
True	False	True	True	False	False
True	True	False	False	True	False
True	True	True	False	False	False

Satisfiability

A statement is **satisfiable** if it has a **model**, that is, a valuation that makes it true.

Statements that never become true are called **unsatisfiable** or **contradictions**.
Statements that are always true and never become false are called **tautologies**.

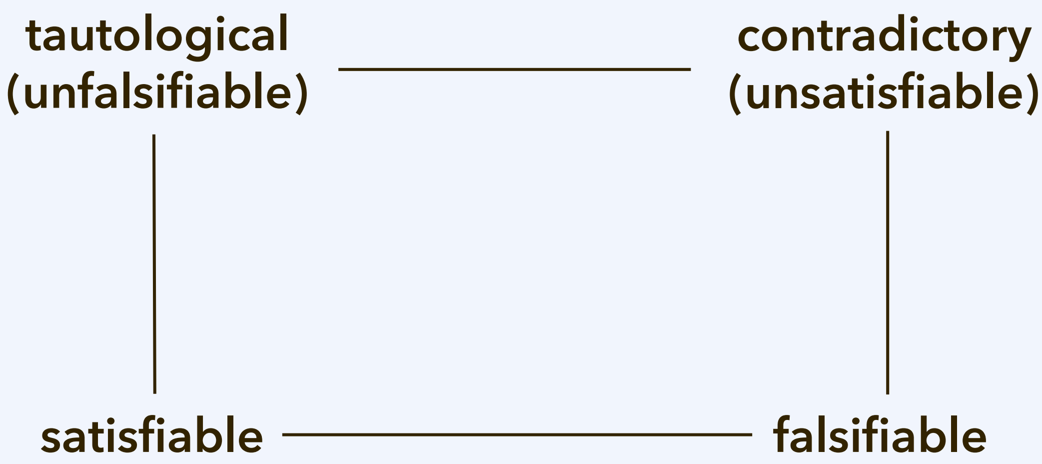
			"p implies q"	*"q implies r"*	*"(p implies q) or (q implies r)"*
p	q	r	$p \to q$	$q \to r$	$(p \to q) \lor (q \to r)$
False	False	False	True	True	True
False	False	True	True	True	True
False	True	False	True	False	True
False	True	True	True	True	True
True	False	False	False	True	True
True	False	True	False	True	True
True	True	False	True	False	True
True	True	True	True	True	True

Satisfiability and the square of opposition

A statement is **satisfiable** if it has a **model**, that is, *a valuation that makes it true*.

Statements that *never become true* are called **unsatisfiable** or **contradictions**.
Statements that are always true and *never become false* are called **tautologies**.

A statement is **falsifiable** if it there is *a valuation that makes it false*.
Statements that are *both satisfiable and falsifiable* are called **contingent**.



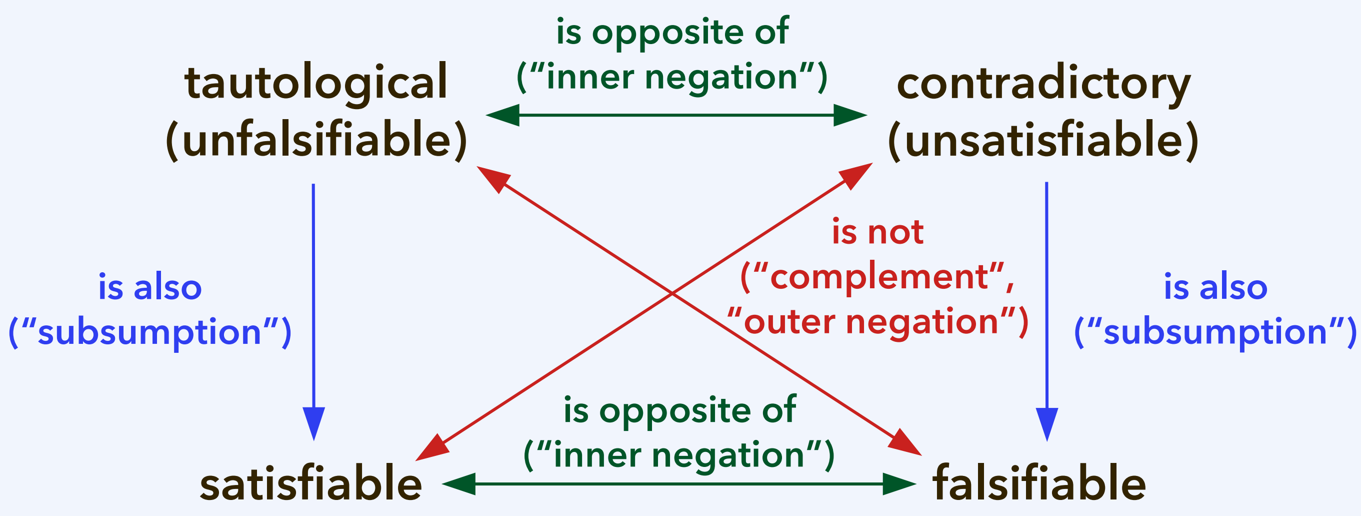
Satisfiability and the square of opposition

If a statement S is tautological, then its negation $\neg S$ is contradictory.

– $(p \rightarrow q) \vee (q \rightarrow r)$ is a tautology; so $\neg((p \rightarrow q) \vee (q \rightarrow r))$ is a contradiction.

The negation of "statement S is satisfiable" is "statement S is a contradiction."

– $(p \vee q) \wedge (\neg p \vee \neg q)$ is satisfiable; therefore it is not a contradiction.

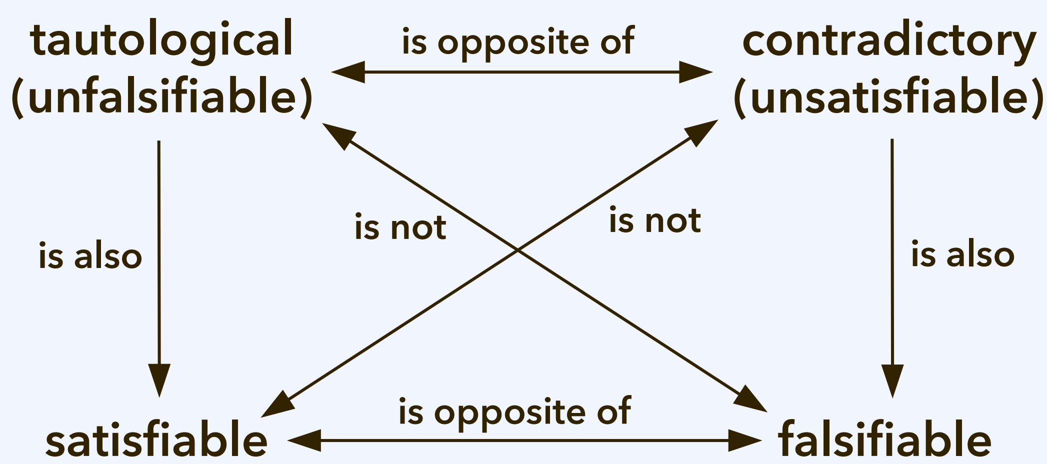


Satisfiability and the square of opposition

Where do the following propositional logic statements belong?

1. $(p \wedge q) \rightarrow p$ *"(p and q) implies p"*
2. $p \leftrightarrow q$ *"p equivalent q"*
3. $\neg(p \vee \neg p)$ *"not (p or not-p)"*

- Are they contradictions (always false) or satisfiable (not always false)?
- Are they tautologies (always true) or falsifiable (not always true)?



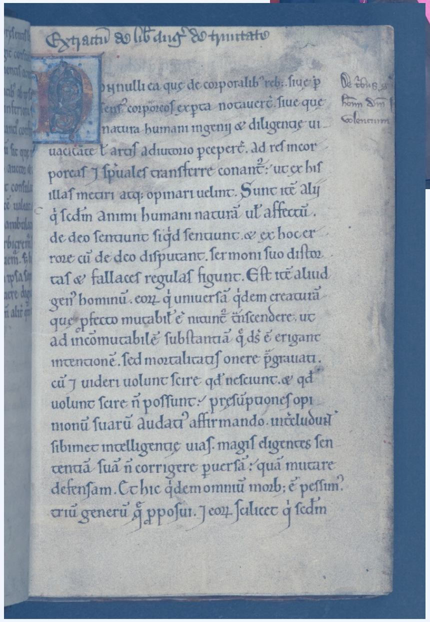
Help! Why logic? … will I ever need this?

You might, if …

- … you will need to formally verify programs or other systems not manually, but by **automated verification** based on model checking;
  - the conditions that are to be verified are usually statements expressed in dedicated kinds of logics (such as temporal logic).
- … you would like to understand all the programming paradigms, not only procedural and object-oriented programming;
  - **logic programming**, for example in PROLOG, is its own paradigm.
- … you plan to develop **semantic web** applications, or if you would like to use non-relational databases;
  - the content of typical non-relational databases is given by statements expressed in a special kind of logic – description logic.

Help! Why logic? … will I ever need this?

… or you might just as well never need it! (Except for the exam.)



University of Central Lancashire
UCLan

CO2412 Computational Thinking

Logical operators
Truth tables

Where opportunity creates success