COMPSCI 250: Introduction to Computation

Lecture #2: Propositions and Boolean Operators
David Mix Barrington and Ghazaleh Parvini
8 September 2023

Propositions and Operators

- What is a Proposition?
- Java Boolean Variables
- Boolean Operators, Compound Propositions
- AND, OR, NOT, and XOR
- Implication and Equivalence
- Tautologies

Previous Clicker Question #3

- Suppose Σ is any non-empty alphabet, and let u and v be any two strings over the alphabet Σ. Which of these statements is not necessarily true?
- (a) if u and v have the same length, then u = v
- (b) if u = v, then u and v have the same length
- (c) if u and v are both empty, then u = v
- (d) if u = v, then every letter of u occurs in v, and every letter of v occurs in u

Not the Answer

Previous Clicker Answer #3

- Suppose Σ is any non-empty alphabet, and let u and v be any two strings over the alphabet Σ. Which of these statements is not necessarily true?
- (a) if u and v have the same length, then u = v
- (b) if u = v, then u and v have the same length
- (c) if u and v are both empty, then u = v
- (d) if u = v, then every letter of u occurs in v, and every letter of v occurs in u

Formal Languages

- Any set of strings over Σ is called a **language** over Σ. We can define languages using any of our ways of defining sets.

- Let $\Sigma = \{0, 1\}$. Define the language X to be all strings that have a 1 at the start, a second different 1 at the end, and 0's in the middle. We can write X as {11, 101, 1001, 10001,...} or as {w: w starts and ends with distinct 1's and has no other 1's}. Later we'll call this language "10^*1".

Decision Problems

- The **decision problem** for a language X is to input a string w (over Σ, the correct alphabet) and return a boolean that is true if $w \in X$ and false if not.
- Given a language, how difficult is it for a computer to solve its decision problem? This is the central question of **formal language theory**. We'll touch on this at the end of the course.

What is a Proposition?

- A **proposition** is a statement that is either true or false.
- In mathematics we want to reason about statements like “$x = 5$” or “these two triangles are congruent” without knowing whether they are true or false. We could say “if $x = 5$, then $x^2 = 25$”, or “if one length and all three angles are the same, then the triangles are congruent”.

More about propositions

- In computing we reason with **assertions** about a program, like “if this method terminates, the value of i is positive”. Ultimately we’d like to say “if the input is as specified, then the output is as specified”, or “the program is **correct**”.
- What isn’t a proposition? Questions, commands, statements without meaning, paradoxes like “this statement is false”, or incompletely specified statements.

Java Boolean Values

- Java has a primitive `boolean` data type, and every boolean has either the value true or the value false.
- We use booleans in the conditions for if or while statements -- if we write "`if (x > 4) y = 5;`", then the statement "`y = 5`" will be executed only if the boolean value "`x > 4`" evaluates to true at run time.

Java Boolean Operators

- The operators ==, !=, >, >=, <, and <= create boolean values from values of other types. We often write methods that return `boolean` values, or use existing `boolean` methods like `equals`. We'll soon see operators that make new booleans from old.

- You may think of a "proposition" as any statement that could be modeled by a boolean variable. Of course, propositions may be about anything, not just computer data.

Making Compound Propositions

- A **compound proposition** is a proposition that is made up from other propositions, called **atomic propositions**, using **boolean operators**.
- If I say “you must have MATH 132, and either CMPSCI 187 or ECE 242”, we can define three atomic propositions and write this as a compound proposition. We let x represent “you have MATH 132”, y be “you have CMPSCI 187, and z be “you have ECE 242”.

Making Compound Propositions

- Now my statement can be written "x, and either y or z". Symbolically, we write this as "$x \wedge (y \vee z)$".
- If x, y, and z are any three `booleans`, the truth of $x \wedge (y \vee z)$ depends on which of x, y, and z are true. In Java, if x, y, and z are `boolean` variables, we can write the expression `x && (y || z)`, and this represents $x \wedge (y \vee z)$.
- This is the **propositional calculus**.

AND and OR

- If x and y are any two propositions, their **conjunction** $x \wedge y$ is the proposition that is true if and only if *both* x and y are true. We read it “x and y”. The Java operators & and && both compute the value of a conjunction -- we usually use && which only evaluates the second argument if it is needed.
- The **disjunction** of x and y is written $x \vee y$, read “x or y”, and is true if either is true, or both. In Java the disjunction is | or ||.

Practice Clicker Question #1

- Let p be "dogs like beef", q be "cats like tuna", and r be "pigs like mud". How would we translate the sentence "Dogs like beef, and either cats like tuna or pigs do not like mud"?
- (a) $(p \wedge q) \vee \neg r$
- (b) $p \wedge \neg(q \wedge r)$
- (c) $p \wedge (q \vee \neg r)$
- (d) $p \vee (q \wedge \neg r)$

Not the Answer

Practice Clicker Answer #1

- Let p be “dogs like beef”, q be “cats like tuna”, and r be “pigs like mud”. How would we translate the sentence “Dogs like beef, and either cats like tuna or pigs do not like mud”?
- (a) $(p \wedge q) \vee \neg r$
- (b) $p \wedge \neg(q \wedge r)$
- (c) $p \wedge (q \vee \neg r)$
- (d) $p \vee (q \wedge \neg r)$

NOT and XOR

- The **negation** of x is written $\neg x$, is read “not x”, and is true when x is false and false when x is true. In Java the negation operator is !.
- The **exclusive or** of x and y is written $x \oplus y$, read “x exclusive or y” or “x or y, but not both”, and is true if one of x and y is true and the other false. In Java we can write “x ^ y” to compute the exclusive or of x and y.

Implication

- The last two boolean operators we will define are **implication** and **equivalence**. These are important in mathematics because each expresses a relationship between propositions that we often want to prove.
- The implication $x \rightarrow y$ is read “if x, then y” or “x implies y”. It is true if *either* x is false or y is true. Equivalently, it is true *unless* x is true and y is false. It’s important to learn this formal definition, whatever you think “if” means.

Practice Clicker Question #2

- Let p be “frogs are green” and q be “trout live in trees”. How would we translate the English sentence “If trout do not live in trees, then frogs are green”?
- (a) $\neg(q \rightarrow p)$
- (b) $\neg q \rightarrow p$
- (c) $p \rightarrow \neg q$
- (d) $\neg q \wedge p$

Not the Answer

Practice Clicker Answer #2

- Let p be "frogs are green" and q be "trout live in trees". How would we translate the English sentence "If trout do not live in trees, then frogs are green"?
- (a) $\neg(q \rightarrow p)$ "It is not the case that if trout live in trees, then frogs are green"
- (b) $\neg q \rightarrow p$
- (c) $p \rightarrow \neg q$ "If frogs are green, then trout don't"
- (d) $\neg q \wedge p$ "Trout don't, and frogs are green"

`false` implies anything

- Normally in mathematics we want to make some **assumptions** and prove that some must be true if the assumptions are true. This is an implication.
- Given our rule, from any false proposition we can prove anything else. Bertrand Russell gave an example of a proof of "I am Elvis" from the premise "$0 = 1$". ("$1 = 2$ by arithmetic, Elvis and I are two people, thus Elvis and I are one person".)

Equivalence

- Two boolean values are **equivalent** if they are both true or both false. If x and y are propositions, $x \leftrightarrow y$ is the proposition that x and y are equivalent. We can write this in Java as x == y.
- We are often interested in the equivalence of two compound propositions with the same atomic propositions. For example, "$x \rightarrow y$" and "$\neg x \vee y$" are equivalent.

More on Equivalence

- How do we know this? They are each true in three of the four possible cases -- they are false only if x is true and y is false. They have the same **truth tables**, as we will soon see.

- As in Java, we have rules for precedence of operations. Negation is first, then the operators $\wedge$, $\vee$, and $\oplus$, then the operators $\rightarrow$ and $\leftrightarrow$. So we can write our equivalence of $x \rightarrow y$ and $\neg x \vee y$ as the single compound proposition $(x \rightarrow y) \leftrightarrow (\neg x \vee y)$.

Tautologies

- Notice a special fact about the compound proposition "$(x \to y) \leftrightarrow (\neg x \vee y)$". It is true in *all four* possible situations of truth values for x and y, so it is *always true*. We call such a compound proposition a **tautology**.
- In the next lecture we will learn a systematic method to show that a compound proposition is a tautology, by checking all the possible combinations of values of its atomic propositions.

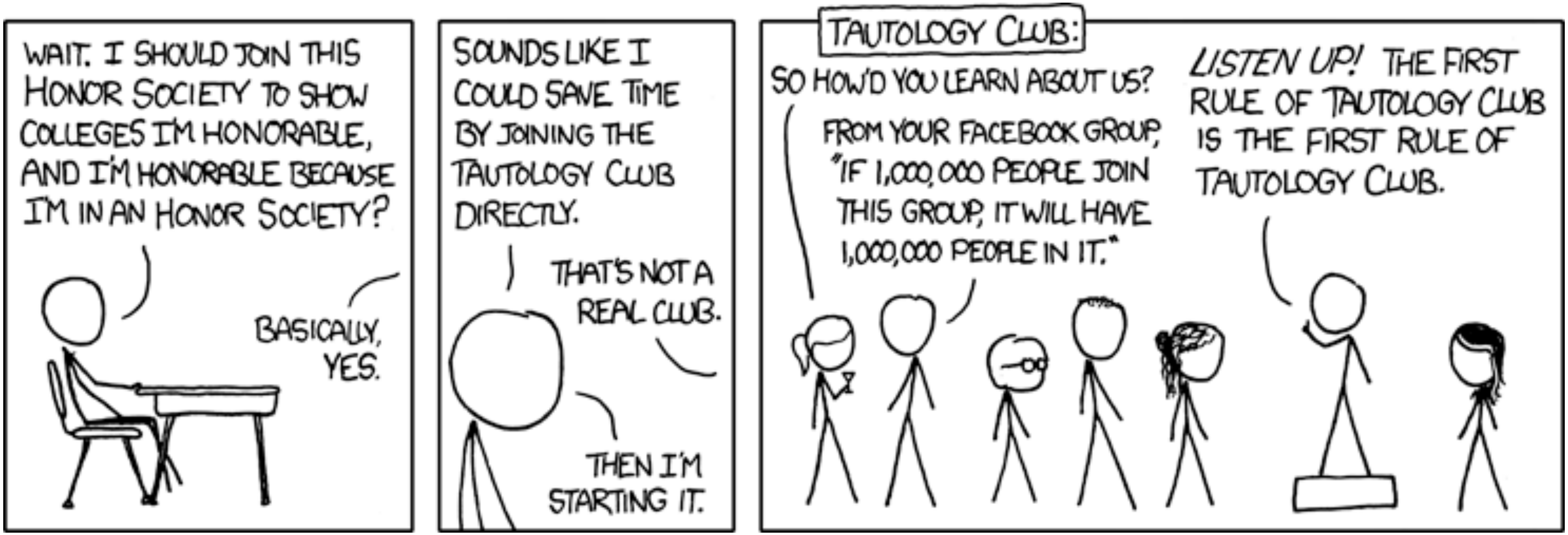
Practice Clicker Question #3

- Which of these statements *is not* a tautology?
- (a) If trout either do or don’t live in trees, then trout don’t live in trees.
- (b) It is not the case that “trout live in trees if and only if trout do not live in trees”.
- (c) If trout live in trees, then it is not the case that trout do not live in trees.
- (d) Either trout live in trees, or trout do not live in trees, but not both.

Practice Clicker Answer #3

- Which of these statements *is not* a tautology?
- (a) If trout either do or don't live in trees, then trout don't live in trees. $(p \vee \neg p) \rightarrow \neg p$
- (b) It is not the case that "trout live in trees if and only if trout do not live in trees". $\neg(p \leftrightarrow \neg p)$
- (c) If trout live in trees, then it is not the case that trout do not live in trees. $(p \rightarrow \neg(\neg p))$
- (d) Either trout live in trees, or trout do not live in trees, but not both. $p \oplus \neg p$

Obligatory xkcd Reference



xkcd.com/703

The Bigger Picture

- Next week we will see how to use particular tautologies as rules, chaining them together to verify larger tautologies without having to check all the possible cases.
- If there are many atomic propositions, this may be the only feasible way to verify the tautology. Remember that if there are k atomic propositions, there are 2^k possible cases!

The Bigger Picture

- In mathematics, our central task with boolean values turns out to be verifying that particular implications or equivalences *are* tautologies.
- Verifying $x \rightarrow y$ means that if we assume x, we may conclude y.
- Verifying $x \leftrightarrow y$ means that x and y are in effect the same compound proposition.