Mathematical Foundations



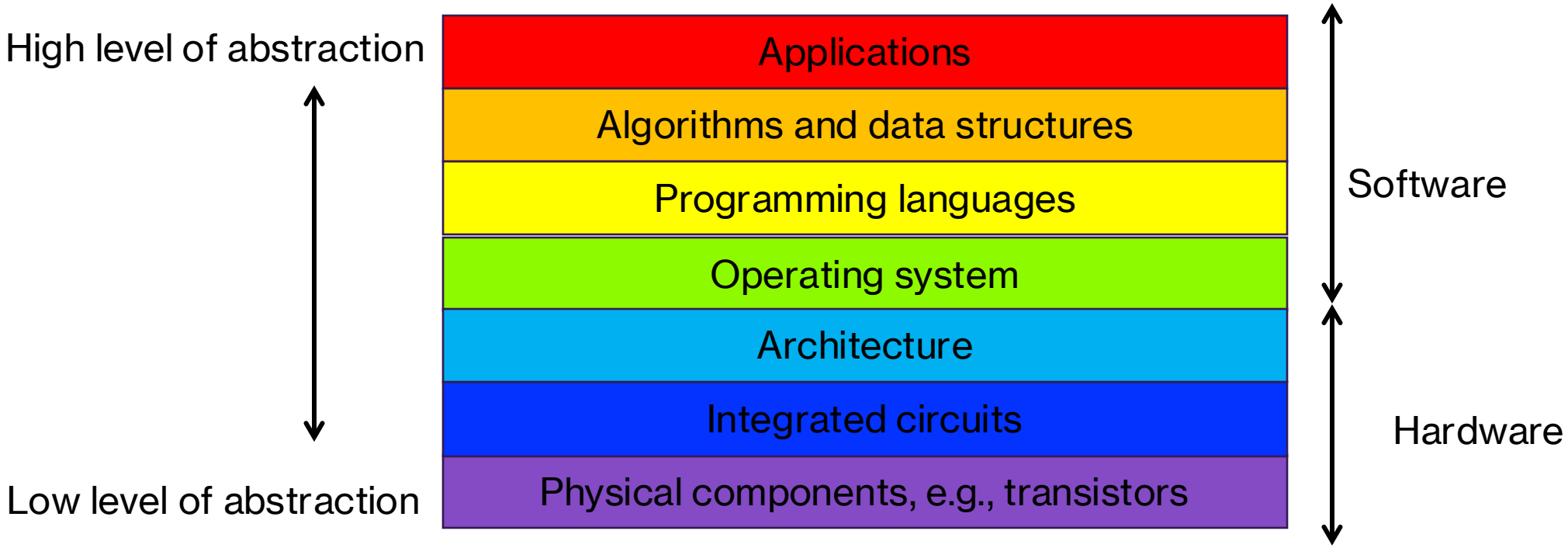
https://xkcd.com/1153/
https://www.explainxkcd.com/wiki/index.php/1153:_Proof

Introduction to Proofs

CS51 – Fall 2025

Abstraction

- Critical technique for managing complexity. We will simplify and generalize information by hiding details when they are not important. This allows us to cultivate a higher level of understanding without being overwhelmed with insignificant details.



Proofs

- **Proofs** are mathematical techniques that will allow us to demonstrate the truth of a claim, starting with certain assumptions and using logical reasoning to reach the conclusion while leaving no doubt to us or the reader of our proof.
  - For example, the claim might have to do about the correctness or efficiency of an algorithm or circuit we designed to solve a problem.
- Proofs have both practical implications as they reduce bugs and theoretical implications as they give us insights into structural aspects of problems we try to tackle.
- Proofs demonstrate that a statement is true for *all possible cases.* This contrasts “good-enough” experimental approaches where we just try a few cases.

Logic

- We are familiar with the concept of truth and falsity, both through Boolean algebra and working with circuits.
- A lot of what we have been learning falls under **Logic**, the study of truth and falsity, of theorems, and proofs.
- Logic is the foundation of all computer science. It's behind the programs we write and behind the gates we connect to build circuits.
- Many types of logic. We will focus on **propositional logic**.

Propositions

- A **proposition** is a statement that is true or not. For example,
  - “CS51 has an enrolment of 47 students.”
  - “Pomona College is the founding college of the Claremont Colleges Consortium.”
- For a particular proposition p, the **truth value** of p is its truth or falsity. The truth values for those statements are false and true, respectively.
- Propositional logic is the study of propositions, including how to formulate statements as propositions, how to evaluate whether a proposition is true or false, and how to manipulate propositions.

Atomic and compound propositions

- An **atomic proposition** is a proposition that is conceptually indivisible. It is also known as a **Boolean variable**. For example:
  - "Pomona College's mascot is Cecil the Sagehen."
- A **compound proposition** is a proposition that is built up out of atomic propositions $p_1, p_2, \ldots, p_k$. It is also called a **Boolean expression** or **Boolean function/formula** over $p_1, p_2, \ldots, p_k$. For example:
  - Claremont is a city in Southern California or Portland is a city in Southern California.

Logical connectives we have seen: $\neg, \wedge, \vee, \oplus$

- **Logical connectives** are the glue that creates the more complicated compound propositions from simpler propositions. We have already seen four:
  - Negation [not, $\neg$]. The proposition $\neg p$ ("not p," called the negation of the proposition p) is true when the proposition p is false, and is false when p is true.
  - Conjunction [and, $\wedge$]. The proposition $p \wedge q$ ("p and q," the conjunction of the propositions p and q) is true when both propositions p and q are true and is false when one or both of p or q is false.
  - Disjunction [or, $\vee$]. The proposition $p \vee q$ ("p or q," the disjunction of the propositions p and q) is true when one or both propositions p or q is true and is false when both p and q are false.
  - Exclusive or [xor, $\oplus$]. The proposition $p \oplus q$ is true when one of p or q is true, but not both. Thus $p \oplus q$ is false when both p and q are true, and when both p and q are false.

New logical connectives: ⇒

- Implication [if/then, ⇒]. It expresses a familiar idea from everyday life, though one that's not quite captured by a single English word.
- Consider the sentence "If you tell me how much money you make, then I'll tell you how much I make."
- The proposition $p \Rightarrow q$ is true when the truth of p implies the truth of q. In other words, $p \Rightarrow q$ is true unless p is true and q is false.
- In the implication $p \Rightarrow q$, the proposition p is called the *antecedent* or the *hypothesis*, and the proposition q is called the *consequent* or the *conclusion.*
- Implication doesn't mean that p caused q. Only that p being true implies that q is true.
  - Or in other words, p being true lets us conclude that q is true.

p	q	$p \Rightarrow q$
T	T	T
$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{F}$
F	T	T
F	F	T

Example: truth values of implications

- Remember, $p \Rightarrow q$ is true unless p is true and q is false.
- The following propositions are all true:
  - 1 + 1 = 2 implies that 2 + 3 = 5. (“True implies True” is true.)
  - 2 + 3 = 4 implies that 2 + 2 = 4. (“False implies True” is true.)
  - 2 + 3 = 4 implies that 2 + 3 = 6. (“False implies False” is true.)
- But the following proposition is false:
  - 2 + 2 = 4 implies that 2 + 1 = 5. (“True implies False” is false.)
  - This is false because 2 + 2 = 4 is true, but 2 + 1 = 5 is false.

p	q	$p \Rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

New logical connectives: ⇔

- If and only if [⇔]: The proposition $p \Leftrightarrow q$ ("p if and only if q") is true when the propositions p or q have the same truth value (both p and q are true, or both p and q are false), and false otherwise.
- The reason that ⇔ is read as "if and only if" is that $p \Leftrightarrow q$ means the same thing as the compound proposition $(p \Rightarrow q) \wedge (q \Rightarrow p)$.

p	q	$p \Rightarrow q$	$q \Rightarrow p$	$(p \Rightarrow q) \wedge (q \Rightarrow p)$	$p \Leftrightarrow q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	T	F	F	F
F	F	T	T	T	T

Precedence of logical connectives

- The **precedence** of our logical connectives is:
  - negation (¬) has the highest precedence;
  - then there is a three-way tie among $\wedge$, $\vee$, and $\oplus$;
  - then there's $\Rightarrow$;
  - then finally $\Leftrightarrow$ has the lowest precedence.

Truth tables for basic logical connectives

p	$\neg p$
T	F
F	T

p	q	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \Rightarrow q$	$p \Leftrightarrow q$
T	T	T	T	F	T	T
T	F	F	T	T	F	F
F	T	F	T	T	T	F
F	F	F	F	F	T	T

Truth tables for more complex propositions

- We can now build truth tables for more complex propositions. For example, to find the truth table for $p \wedge q \Rightarrow \neg q$ we have:

p	q	$p \wedge q$	$\neg q$	$p \wedge q \Rightarrow \neg q$
T	T	T	F	F
T	F	F	T	T
F	T	F	F	T
F	F	F	T	T

- Which shows that the given proposition is true precisely when at least one of p and q is false.

Practice time

- Cecil Sagehen, who is 47 years old, was asked at the doctor's office: "If you are over 55 years old, do you have an advance care directive?"
- What should she answer?

Practice time

- She should answer “yes”:
- p = "over 55 years old"
- q = "have an advance care directive"
- p ⇒ q is true whenever p is false, and “over 55 years old” is false for her. (That is, False ⇒ anything is true.)

Tautologies

- A proposition is a **tautology** if it is true under *every* truth assignment.
- One reason that tautologies are important is that we can use them to reason about logical statements, which can be particularly valuable when we're trying to prove a claim.
- For example, if we had a compound proposition over the propositions p and q, it would be a tautology because it is true under all possible assignments for p and q.

p	q	compound proposition
T	T	T
T	F	T
F	T	T
F	F	T

Practice time: Important tautologies

- *Law of the excluded middle*. for any proposition p, either p is true or p is false; there is nothing "between" true and false. This means that $p \vee \neg p$ is a tautology.
- *Modus ponens*: $[p \wedge (p \Rightarrow q)] \Rightarrow q$. If we know both that (a) p is true and that (b) the truth of p implies the truth of q, then we can conclude that q is true.
- *Modus tolens*: $[\neg q \wedge (p \Rightarrow q)] \Rightarrow \neg p$. If we know both that (a) q is false and that (b) the truth of p implies the truth of q, then we can conclude that p is false.

Answer: Important tautologies

			Law of excl mid			**Modus ponens**			**Modus tolens**
p	q	$\neg p$	$\boldsymbol{p \vee \neg p}$	$p \Rightarrow q$	$p \wedge (p \Rightarrow q)$	$[\boldsymbol{p} \wedge (\boldsymbol{p} \Rightarrow \boldsymbol{q})] \Rightarrow \boldsymbol{q}$	$\neg q$	$\neg q \wedge (p \Rightarrow q)$	$[\neg \boldsymbol{q} \wedge (\boldsymbol{p} \Rightarrow \boldsymbol{q})] \Rightarrow \neg \boldsymbol{p}$
T	T	F	$\boldsymbol{T}$	T	T	$\boldsymbol{T}$	F	F	$\boldsymbol{T}$
T	F	F	$\boldsymbol{T}$	F	F	$\boldsymbol{T}$	T	F	$\boldsymbol{T}$
F	T	T	$\boldsymbol{T}$	T	F	$\boldsymbol{T}$	F	F	$\boldsymbol{T}$
F	F	T	$\boldsymbol{T}$	T	F	$\boldsymbol{T}$	T	T	$\boldsymbol{T}$

Satisfiable propositions

- A proposition is **satisfiable** if it is true under at least one truth assignment.
- A proposition is **unsatisfiable** if it is not satisfiable.
- For example, the proposition $(p \Leftrightarrow q) \wedge (p \oplus q)$ is unsatisfiable.

p	q	$p \Leftrightarrow q$	$p \oplus q$	$(p \Leftrightarrow q) \wedge (p \oplus q)$
T	T	T	F	$\boldsymbol{F}$
T	F	F	T	$\boldsymbol{F}$
F	T	F	T	$\boldsymbol{F}$
F	F	T	F	$\boldsymbol{F}$

Practice time: Satisfiable or not?

- Is the proposition $p \vee q \Rightarrow \neg p \wedge \neg q$ satisfiable?

Practice time: Satisfiable or not?

- Is the proposition $p \vee q \Rightarrow \neg p \wedge \neg q$ satisfiable?
  - *Yes, it is.*

p	q	$p \vee q$	$\neg p$	$\neg q$	$\neg p \wedge \neg q$	$p \vee q \Rightarrow \neg p \wedge \neg q$
T	T	T	F	F	F	F
T	F	T	F	T	F	F
F	T	T	T	F	F	F
F	F	F	T	T	T	T

Complexity theory

- Complexity theory is the subfield of computer science devoted to understanding the computational resources (e.g., time and memory) necessary to solve particular problems.
- One of the central problems is the SAT problem (Boolean Satisfiability problem): you are given a proposition ϕ over n variables and asked to determine whether ϕ is satisfiable.
- The problem is pretty simple to solve. For example, you construct the truth table for the proposition and then check whether there are any True rows in the ϕ's column.
- But this algorithm is not very fast because the truth table has 2^n rows. Even a small n means that this algorithm will not terminate in our lifetime. E.g., for 2^{300} we would exceed the number of particles in the known universe!
- So although there is an algorithm that solves the SAT problem, it is unclear whether there is a substantially more efficient algorithm to solve it.

Satisfiability and a million dollars prize

- This problem is so big that the Clay Mathematics Institute included it in the seven Millennium Prize Problems and will give a $1 million prize to anyone who solves it.
- Why do we care so much about this problem?
- SAT is just as hard as many other computational problems that belong in the class of NP. For NP problems(non-deterministic polynomial time), it is easy to **verify** the correct answer (the correct answer can be verified in polynomial time).
- The question is whether these problems can also be **solved** in polynomial time. This class of problems is known as P.
- That is, we do not know whether P=NP.
- It is widely believed that P≠NP. Solving this problem would have profound implications on fields like cryptography.

Complexity theory pioneers

- **Cook-Levin Theorem**: If you can solve SAT efficiently, then you can solve any problem in NP efficiently.
- Karp also contributed to theory of NP-completeness and along with Lipton showed a relationship between SAT and circuits that have a polynomial number of logic gates



Stephen Cook



Leonid Levin



Richard Karp

Tautology and satisfiability

- Let φ be any proposition. Then φ is a tautology exactly when $\neg\varphi$ is unsatisfiable.
- φ is a tautology when the truth table for φ is all trues, which happens exactly when the truth table for $\neg\varphi$ is all falses. And that's precisely the definition of $\neg\varphi$ being unsatisfiable!

φ	$\neg\varphi$
T	F
T	F
T	F
T	F

Logical equivalence

- Two propositions φ and ψ are logically equivalent, written $\varphi \equiv \psi$, if they have exactly identical truth tables (in other words, their truth values are the same under every truth assignment).
- To state it differently: φ and ψ are logically equivalent whenever $\varphi \Leftrightarrow \psi$ is a tautology.
- For example, $\neg(p \wedge q)$ and $(p \wedge q) \Rightarrow \neg q$ are logically equivalent:

p	q	$p \wedge q$	$\neg(\boldsymbol{p \wedge q})$	$\neg q$	$(p \wedge q) \Rightarrow \neg q$
T	T	T	$\boldsymbol{F}$	F	$\boldsymbol{F}$
T	F	F	$\boldsymbol{T}$	T	$\boldsymbol{T}$
F	T	F	$\boldsymbol{T}$	F	$\boldsymbol{T}$
F	F	F	$\boldsymbol{T}$	T	$\boldsymbol{T}$

Literals, CNFs, and DNFs

- A **literal** is a Boolean variable (a.k.a. an atomic proposition) or the negation of a Boolean variable.
  - This means that both p and $\neg p$ are literals.
- A proposition is in conjunctive normal form (CNF) if it is the conjunction of one or more clauses, where each clause is the disjunction of one or more literals.
- A proposition is in disjunctive normal form (DNF) if it is the disjunction of one or more clauses, where each clause is the conjunction of one or more literals.
- Less formally, a proposition in conjunctive normal form is “the and of a bunch of ors,” and a proposition in disjunctive normal form is “the or of a bunch of ands.”

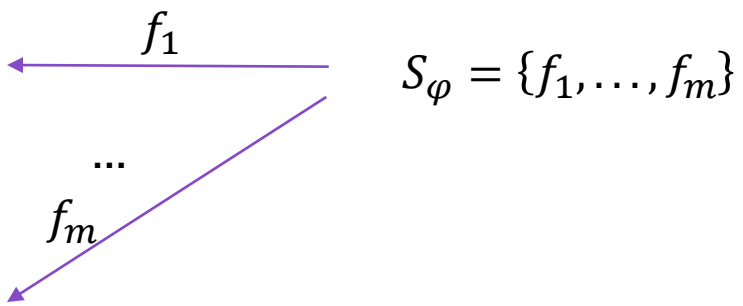
A somewhat familiar theorem

- *Theorem 1*: For any proposition φ, there is a proposition ψ_{dnf} over the same Boolean variables and in disjunctive normal form such that $\varphi \equiv \psi_{dnf}$.

Proof for Theorem 1

- Let φ be an arbitrary proposition, say over the Boolean variables $p_1, \ldots, p_k$.
- Build the truth table for φ, and let $S_\varphi = \{f_1, f_2, \ldots, f_m\}$ denote the set of m truth assignments for $p_1, \ldots, p_k$ under which φ is true.

p_1	p_2	...	p_k	φ
T	F	...	F	T
T	F	...	T	F
...	...	...	...	...
F	F	...	T	T



Proof for Theorem 1

- Looking at the truth table of φ, for any particular assignment f_i for the variables $p_1, \ldots, p_k$ that makes φ true, we'll construct a conjunction c_{f_i} that's true under f_i and false under all other assignments that makes φ true.
  - Let $x_1, x_2, \ldots, x_l$ be the variables assigned true by f_i, and $y_1, y_2, \ldots, y_{k-l}$ be the variables assigned false by f_i. Then the clause:
  - $c_{f_i} = x_1 \wedge x_2 \wedge \ldots \wedge x_\ell \wedge \neg y_1 \wedge \neg y_2 \wedge \ldots \wedge \neg y_{k-l}$ is true under f_i, and c_{f_i} is false under every other truth assignment.

p_1	p_2	...	p_k	φ
T	***F***	...	***F***	***T***
T	F	...	T	F
...	...	...	...	...
F	F	...	T	T

$$c_{f_1} = p_1 \wedge \neg\, p_2 \wedge \cdots \wedge \neg p_k$$

Proof for Theorem 1

- We can now construct a DNF proposition ψ_{dnf} that is logically equivalent to φ by "or"ing together the clause c_{f_i} for each truth assignment $f_i, \mathrm{i} = 1, \ldots, \mathrm{m}, m \geq 1$ that makes φ true.
- Define $\psi_{dnf} = c_{f_1} \vee c_{f_2} \vee \ldots \vee c_{f_m}$ (*)

p_1	p_2	...	p_k	φ	ψ_{dnf}
T	***F***	...	***F***	***T***	***T***
T	F	...	T	F	F
...	...	...	...	...	...
F	***F***	...	***T***	***T***	***T***

$$\psi_{dnf} = c_{f_1} \vee \cdots \vee c_{f_m}$$

Proof for Theorem 1

- Then ψ_{dnf} is true under every truth assignment f_i under which φ was true (because the clause c_{f_i} is true under f_i).
- And, for a truth assignment f under which φ was false, every disjunct in ψ_{dnf} evaluates to false, so the entire disjunction is false under such an f, too.
- Thus, $\varphi \equiv \psi_{dnf}$
- Are we done?

p_1	p_2	...	p_k	φ	ψ_{dnf}
T	F	...	F	$\boldsymbol{T}$	$\boldsymbol{T}$
T	F	...	T	$\boldsymbol{F}$	$\boldsymbol{F}$
...	...	...	...	...	...
F	F	...	T	$\boldsymbol{T}$	$\boldsymbol{T}$

$$\psi_{dnf} = c_{f_1} \vee \cdots \vee c_{f_m}$$

$$\psi_{dnf} = c_{f_1} \vee c_{f_2} \vee \ldots \vee c_{f_m} \ (*)$$

Proof for Theorem 1

- There's one thing we have to be careful about: what happens if S_φ is an empty set, that is if φ is unsatisfiable?
- The construction in $(*)$ doesn't work, but it's easy to handle this case separately: we simply choose an unsatisfiable DNF proposition like $p \wedge \neg p$ as ψ_{dnf}.

p_1	p_2	…	p_k	φ
T	F	…	F	F
T	F	…	T	F
…	…	…	…	…
F	F	…	T	F

Another familiar theorem

- Theorem 2: All propositions are expressible in CNF. For any proposition ϕ, there is a proposition φ_{cnf} over the same Boolean variables and in conjunctive normal form such that $\varphi \equiv \varphi_{cnf}$.

Proof for Theorem 2

- If φ is a tautology, the task is easy; just define $\varphi_{cnf} = p \vee \neg p$.

p_1	p_2	…	p_k	φ
T	F	…	F	T
T	F	…	T	T
…	…	…	…	…
F	F	…	T	T

Proof for Theorem 2

- Otherwise, φ is a non-tautology, say over the variables $p_1, \ldots, p_k$. Using Theorem 1 we can construct a DNF proposition ψ that is logically equivalent to $\neg\varphi$ of the form:
- $\psi = \left(c_1^1 \wedge \cdots \wedge c_k^1\right) \vee \left(c_1^2 \wedge \cdots \wedge c_k^2\right) \vee \cdots \vee (c_1^m \wedge \cdots \wedge c_k^m)$, for some $m \geq 1$ where each where each c_i^j is a literal.
- Recall that $\psi \equiv \neg\varphi$, thus $\neg\psi \equiv \varphi$.

p_1	p_2	...	p_k	φ	$\neg\varphi$	ψ
T	F	...	F	T	F	F
T	F	...	T	F	T	T
...	...	...	...	...	...	...
F	F	...	T	T	F	F

Proof for Theorem 2

- We will negate ψ and use De Morgan's Laws: $\neg(p \vee q) \equiv \neg p \wedge \neg q$ and $\neg(p \wedge q) \equiv \neg p \vee \neg q$:
- $\neg\,\psi = \neg[(c_1^1 \wedge \cdots \wedge c_k^1) \vee (c_1^2 \wedge \cdots \wedge c_k^2) \vee \cdots \vee (c_1^m \wedge \cdots \wedge c_k^m)] \equiv$ $[(\neg c_1^1 \vee \cdots \vee \neg c_k^1) \wedge (\neg c_1^2 \vee \cdots \vee \neg c_k^2) \wedge \cdots \wedge (\neg c_1^m \vee \cdots \vee \neg c_k^m)]$.
- But this expression is in CNF! Thus, we've constructed a proposition in conjunctive normal form that's logically equivalent to $\neg\psi \equiv \varphi$.

p_1	p_2	...	p_k	φ	$\neg\varphi$	ψ	$\neg\boldsymbol{\psi}$
T	F	...	F	T	F	F	T
T	F	...	T	F	T	T	F
...	...	...	...	...	...	...	...
F	F	...	T	T	F	F	T

Proofs in computer science

- The type of proof we saw for Theorem 1 and 2 are known as **proof by cases**, with two cases corresponding to φ being satisfiable or a tautology and φ being unsatisfiable or a non-tautology being proven separately.
- There are many types of proofs used in computer science.
  - Together, we will see loop invariant proofs and proofs by induction but if you continue with more advanced computer science courses you will encounter even more.
- There's genuine creativity in producing proofs along with key strategies that one can learn.