Formal Proof

Understanding, writing and evaluating proofs

David Megías Jiménez (coordinador)
Robert Clarisó Viladrosa

Amb la col·laboració de
M. Antònia Huertas Sánchez

PID_00154272

Table of contents

Introduction 5

Goals 6

1. Defining formal proofs 7

2. Anatomy of a formal proof 10

3. Tool-supported proofs 13

4. Planning formal proofs 17
- 4.1. Identifying the problem 18
- 4.2. Reviewing the literature 18
- 4.3. Identifying the premises 19
- 4.4. Understanding the problem 20
- 4.5. Formalising the problem 20
- 4.6. Selecting a proof strategy 21
- 4.7. Developing the proof 22
- 4.8. Reevaluating the proof 24
- 4.9. Improving readability 25

5. Proof strategies 27

6. The Internet and formal proofs 35

7. Examples of proofs in IS and computing 36

8. Evaluating formal proof research 38

9. Further reading 40

Summary 41

Self-assessment exercises 42

Solutions 43

Glossary 45

Bibliography 46

Introduction

In this module, we present formal proofs, a research strategy with applications to several research fields in computing, engineering and exact sciences in general.

Formal proofs use known facts and the deduction rules of logic to reach conclusions. As proofs are based on logics and mathematics, the conclusion of a proof is completely certain. Thus, proofs can be used to confirm hypothesis and conjectures with complete assurance.

This module presents the goals of formal proof and their role in computing research. We discuss the notation, vocabulary and structure of a proof, as well as general proof strategies that can be useful in different problems. We also consider software tools that provide support in the development of formal proofs. Finally, we discuss the evaluation criteria that can used to assess the quality of a proof.

In some sections, we work on specific examples of proofs. Although the required background and notation is introduced as needed, a basic knowledge of mathematics, logic and computer science concepts will be helpful to the reader.

Goals

1. To learn what a formal proof is and the type of research problems in which it is valuable.

2. To understand the structure and vocabulary used in formal proofs.

3. To understand the process of writing a proof and the different decisions that must be taken.

4. To become familiar with different proof strategies and understand how they can be applied.

5. To identify different software tools that can help in the automation of formal proofs.

6. To locate the Internet resources that can be used in the construction of a proof.

7. To learn how to evaluate formal proofs critically and how they relate to other research strategies.

1. Defining formal proofs

Formal proof is a well-established research approach that dates back to the origins of mathematics in ancient Greece. The goal of formal proofs is to provide certainty about the validity of a statement through rigorous deduction. Starting from the definition of the problem and the set of accepted axioms, a proof follows the logical inference rules until the statement is validated or refuted. Nowadays, proofs are used in many theoretical disciplines, including logics, mathematics, physics and computing.

Let us consider a hypothetical example:

Alice is a security researcher who has developed a new cryptographic algorithm to cipher messages. One desirable property of such algorithms is that it should be hard or impossible to recover the original message (plain-text) from the encoded message (cipher-text) without knowing the encryption key.

This property is critical, as the algorithm may be used to transfer important information that should be kept private. Testing the algorithm is not sufficient for Alice, as she might miss a corner case in which security can be breached. Therefore, she will need to prove that the algorithm is secure in every potential scenario. In this proof, she can use established results from the fields of information theory, cryptography and mathematics.

The core asset of formal proofs lies in its *formal* nature. Mathematical language establishes the meaning of a statement in a precise and unambiguous way. Logic defines the catalogue of inference rules that can used to draw conclusions from previous statements. Therefore, a proof lets us claim the conclusions with complete confidence.

However, formality comes with a price: a loss in expressiveness. Formal languages define strict rules for building new statements. Therefore, there may be some statements that do not have a representation in that language. Selecting an adequate logic to formalise the problem is part the process of developing a proof.

Furthermore, the effort required to formalise a problem exceeds that of describing it in natural language. Natural language can deal with ambiguity and

implicit knowledge, while in a formal setting all knowledge must be stated explicitly, including commonsensical information.

Therefore, formal proofs do not make sense in every computing context: many important research questions can be investigated through an empirical quantitative or qualitative evaluation. Also, most systems do not require the degree of assurance provided by a formal proof. In these cases, robust design and development methods and thorough testing provide sufficient confidence. However, there are specific domains and research problems in which formal proofs are a vital asset, such as:

- General laws that govern nature, mathematics and computing.
- Methods, algorithms and architectures for solving a problem in an abstract and general setting.
- Hardware and embedded designs that are either (a) expensive to fabricate, (b) mass-produced, (c) difficult to replace or (d) all the above.
- Safety-critical systems whose failure may lead to irreparable harm, *e.g.* systems in the following sectors: health, aerospace, automotive, energy, military, telecommunication, economy, ...

Table 1 illustrates examples of research fields and research questions in computing in which a formal proof may be required. This list is not meant to be exhaustive.

In addition to the problem being studied, a formal proof can be classified according to its style. There are different ways to develop a formal proof:

- Writing a pen-and-paper proof that can be understood and reproduced by other researchers. This is the "classical" approach to proofs in mathematics, logic and sciences. Correctness, readability and elegance are the usual criteria to evaluate this type of proofs.
- Formalising the problem as a statement in a logic theory in which it can be proved by a theorem prover. Theorem provers can apply inference rules automatically, simplifying the mechanical steps of the proof. In this approach, the main evaluation criteria are whether the theorem prover can complete the proof and, in some cases, the resources (execution time, memory usage) required by the theorem prover to validate the statement.
- Constructing a hybrid proof in which some subproblems are proved in the pen-and-paper style and others are proved with the help of theorem provers. This approach takes advantage of the best of both worlds: the creativity of pen-and-paper tools and the automation of tedious tasks from tool-supported proofs.

Theorem provers...

...are software tools capable of reasoning in some logic, either automatically or through user interaction. Section 3 provides additional information on automated and interactive theorem provers.

Table 1

Research field	Research question	
	Problem name	**Statement**
Any (Computing, Math, ...)	Universality ($\forall$)	Do all entities of a given type satisfy a property P?
	Existence ($\exists$)	Can we ensure that there is always *at least one* entity satisfying a property P?
	Uniqueness	Can we guarantee that there is always *exactly one* entity satisfying a property P?
	Implication	Can we certify that property B is a consequence of property A, *i.e.* B is true whenever A is true? ($A \rightarrow B$)
	Equivalence (if-and-only-if)	Can we ensure that properties A and B are equivalent, *i.e.* A is true if B is true and A is false if B is false? ($A \leftrightarrow B$).
Cryptography	Security	Is it possible to recover information about the original message from the encrypted text?
	Authenticity	Can a malicious third party replace the contents of a message without being detected?
	Anonymity	Can a third party learn who is the sender of a message?
	Non-repudiation	Is it possible to certify that an entity sent or received a message?
Calculability	Decidability	Is it possible to write a program that solves a problem P?
	Reduction	Is solving a problem P_1 as hard as solving a problem P_2?
Algorithmics	Termination	Does an algorithm A terminate (converge) after a finite number of steps? Does it happen for all possible inputs?
	Complexity	What is the best/worst/average execution time and memory usage of an algorithm A?
	Determinism	Does the algorithm A always compute the same result given the same input data?
	Correctness	Is the result of an algorithm A correct according to our goal? (*I.e.* does the result always satisfy a desired property?)
	Optimality	Is the result of an algorithm A optimal according to some criteria? (*E.g.* the lowest cost, the highest payoff, the smallest/largest size, ...)
	Approximation	Is the result of an algorithm A always within some distance of the optimal result?
	Equivalence	Is the output of an algorithm A_1 always the same as the algorithm A_2 for the same input data?
Concurrent and distributed systems	Deadlock	Can the system become stalled because a process is waiting for a second process that is in turn waiting for the first one?
	Robustness	Is the system failure-tolerant, *i.e.* can it continue operating even if N of its nodes stop operating or behave maliciously?
	Mutual exclusion	Can a process access a resource or perform a set of actions in isolation from the rest of processes?
Safety-critical systems	Formal verification	Does the implementation of a software or hardware system satisfy its specification?
	Absence of run-time errors	Can we ensure that a program will run without run-time exceptions or errors? (divide by zero, memory leaks, dangling pointers, buffer/stack/integer overflows, round-off errors, ...)
	Safety	Is it possible to ensure that a system will not reach an incorrect state from the initial configuration?
	Liveness	Is it possible to ensure that a system will eventually reach a desired state or perform a desired action from any configuration?
	Consistency	Is the specification of the system consistent, *i.e.* are all views of the system non-contradictory among them? Is each view of the system realizable?

Tool-supported proofs are briefly discussed in Section 3, but in the remainder of this module, we focus on pen-and-paper style proofs.

A factor that is sometimes ignored or left for last-minute tuning is the *audience* of the proof. Readers should be able to understand the problem, the premises, the definitions and the notation, and to follow all the deduction process. This means that we should take into account the readers from the start: their background, the vocabulary they commonly use to talk about the problem and the mathematical notation they are familiar with. All this information should influence our choice of notation, the level of detail and how we engineer our proof. We should make a continuous effort to make the proof comprehensible, using examples, figures and clarifications where it is necessary.

2. Anatomy of a formal proof

Formal proofs are usually divided into three blocks: *preliminaries*, *problem statement* and *proof development*. We illustrate this structure with an example in which the specific details of the proof development have been omitted:

Definition 1 (Queen): A queen is a chess piece that threatens any piece of the board located on the same row, column or diagonal.

Definition 2 (N-Queens): Given an empty $N \times N$-chess board, is it possible to place N queens in such a way that no pair of queens threaten each other?

Preliminaries

Theorem 1 (Feasibility of N-Queens): The N-Queens problem has a solution for any $N \geq 4$.

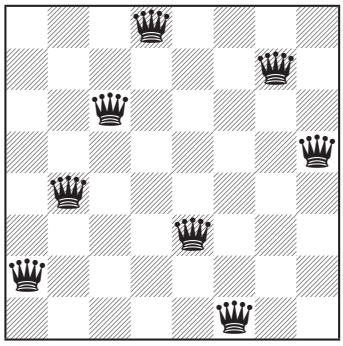
Problem statement

Proof: Let us consider the following algorithm for generating a solution to the N-Queens problem for an arbitrary N: ... This algorithm ensures that there is no pair of queens on the same row and column, but we still have to make sure that there are no queens on the same diagonal. We prove this ... ■

Proof development

The N-Queens Problem

This mathematical puzzle is a generalisation of the 8-Queens problem, initially proposed in 1848. This problem has gathered interest from mathematicians and computer scientists, and it is often used a benchmark for measuring the efficiency of theorem provers and constraint solvers.



A solution to the 8-Queens Problem.

The **preliminaries** introduce the background information necessary to understand the problem and the proof. Here, it is necessary to provide precise definitions of the vocabulary and mathematical notation that will be used to reason about our problem. Finally, we may remind previously proved results that will be used within the proof, citing the original sources.

Sometimes definitions can be provided within the text. This is recommended when the concept being defined is simple or already familiar to the reader, it does not involve complex mathematical notation and it does not need to be referenced in the future. In any other case, each definition should be placed in a separate paragraph labelled as "Definition N" to allow future reference, *e.g.* "this is a conclusion of definition N".

The **problem statement** provides a description of the property we are trying to prove. The information provided in the preliminaries should allow this definition to be clear, concise, precise and unambiguous.

If there are several proofs in the manuscript, the problem statement will also be numbered, *e.g.* "Theorem 7". The term used to classify our statement (*e.g.* "theorem", "lemma" or "claim") is significant as it indicates the goal of our proof within the overall contribution. A *theorem* is a main result to be proved, while a *lemma* is an intermediate result that will be useful in more complex proofs. The term *proposition* may also be used to denote a property that needs to be proved, either a theorem or lemma. An *invariant* is a property that is satisfied in every potential configuration of a dynamic system, *e.g.* a condition that holds in each iteration of a loop. A property that will be proved much later may be introduced as a *claim* or *hypothesis*. These terms may also be used to describe a property that remains unproven, although, in this situation, the term *conjecture* is usually preferred. A *corollary* is a conclusion of a previous theorem or lemma.

Finally, the **proof development** is the core of the proof: a sequence of logical deductions that lead us to conclude our problem statement.

There are several different approaches to complete a proof, *i.e. proof strategies* and some may be more suitable than others for some type of properties. For example, if we are trying to prove that "there is somebody who does not have a pet tiger", a possible line of work could be using a counting argument: proving that there are more humans than tigers (assuming that two humans cannot share a pet and that only humans have pets). Formal proofs typically begin by describing the selected proof strategy.

During the proof development, we may refer to information provided in the preliminaries (definitions, previous results) and other proofs (auxiliary lemmas). Any other step of the proof should infer a new statement from the previously deduced statements (*premises*). It is important to refer, at every time, which premise or preliminary is being used to reach a conclusion and to provide an intuition on how this new conclusion brings us closer to our problem statement.

Finally, we should make sure to signal not only the start but also the end of the proof. It can be either with a sentence like "this completes the proof" or "we have reached our conclusion". Others prefer to use the acronym *q.e.d.*

or "*quod erat demonstrandum*", latin for "this that was to be proved", which is the classical formula for concluding mathematical proofs. Another possible end-of-proof mark is the mathematical *tombstone* or *Halmos* symbol, either ■ or □.

Elementary, my dear Watson

In the proofs provided as example within this module, we indicate the end of the proof with the Halmos symbol (■). This end-of-proof mark was introduced by the mathematician Paul Halmos (1916-2006).

3. Tool-supported proofs

The mechanisation of logic and reasoning has been a long-standing goal in the fields of logic, mathematics and computing. Problems are formalised in a *logic theory* that defines what is a statement (*e.g.* syntax rules like "true is a statement" or "if A is a statement, then $\neg A$ is also a statement") and the set of initially accepted premises called *axioms* (*e.g.* $\neg$false = true,$\neg\neg x = x$). Programs that are able to reason on a specific logic theory are called *theorem provers*. Given a set of *premises* encoded in the syntax of the logic theory and another statement (the *theorem*), the theorem prover checks whether the theorem can be deduced from the premises. Theorem provers can be used as a complement or even a substitute of pen-and-paper proofs.

Unfortunately, there are theoretical results that establish limitations regarding which questions can be answered in a specific logic theory. For example, some logic theories are *undecidable*, *i.e.* it is not possible to automate the decision procedure that checks whether a theorem is the logical consequence of the premises. These theoretical limitations manifest in the classification of theorem provers as *automated* or *interactive*.

Liar paradox

It is not possible to decide whether the statement "This statement is false" is true or false: if it is true, it must be false according to its definition, and vice versa. This is an example of problems arising from self-references in logic.

Automated theorem provers (ATP) are based on decision procedures that are completely automatic: once the user encodes the premises and the theorem to be proved in the syntax of the logic theory, the tool proceeds until it proves the theorem or is able to justify that it cannot be proved. The answer of the tool is simply "Yes", "No" or "I don't know" (if the logic theory is undecidable) and optionally a trace lets the user understand how this conclusion was reached. The core design principle is that there is no user intervention to guide the deduction process during the proof, although some degree of parameterisation or heuristic selection can usually be performed prior to invoking the theorem prover.

In contrast, interactive theorem provers (ITP) or *proof assistants* require user guidance to complete the proof. ITPs are used for logic theories in which there is no automated decision procedure or in which ATPs are inefficient and experts can make informed decisions that are beyond the capabilities of an ATP. ITPs are usually built on top of ATPs that mechanise the parts of the proof that can be automated, and user interaction is only required when the ATP fails to continue. When it requires user assistance, the ITP informs the user about the statement that it is unable to prove, possibly a part of the final theorem or an intermediate statement. Then, the user needs to provide information about how to proceed, for example by identifying other statements that should be proved first, adding additional premises or suggesting a proof strategy (*e.g.* "use proof by exhaustion and consider the following cases"). This

Complementary readings

MacKenzie D. (1995) *The Automation of Proof: A Historical and Sociological Exploration*. IEEE Annals of the History of Computing. 17(3):7–29.
Hales T. C (2008) *Formal Proof*. Notices of the AMS, 55(11):1370–1380.
Bundy A. (1999) *A Survey of Automated Deduction*. Artificial Intelligence Today, LNCS 1600, 153–174.
Geuvers H. (2009) *Proof Assistants: History, Ideas and Future*. Sadhana, 34(1):3–25.

type of interaction requires a high-level of expertise in order to understand the information provided by the tool, decide the most likely path to success and communicate these instructions to the tool.

Should we use a theorem prover?

The answer to this question depends on the *size* of the problem. Using a theorem prover requires some effort from the part of the user, for example, to encode the statements in the underlying logic theory. For example, it is also necessary to explicitly formalise domain-specific knowledge and "common sense" knowledge about the problem. This formalisation effort is *significant* for somebody who is not familiar with logic terminology and notation. Therefore, for simple and small problems, the use of a theorem prover may be overkill.

Proof trivia

In the Metamath proof language (`www.metamath.org`), proving that 2+2=4 using only the fundamental axioms of arithmetic requires 25,933 deduction steps.

For example, if we are trying to prove the correctness of a short and abstract algorithm, the pen-and-paper style might be sufficient for our goal. On the contrary, the formal verification of a program with 100,000 lines of code will definitely require the use of automated or interactive theorem provers.

Selecting the right theorem prover

From our point of view as users of theorem provers, automation is a very desirable trait. Therefore, the general rule of thumb is "whenever possible, choose an ATP instead of an ITP". Another rule of thumb caused by practical considerations is "use the most efficient ATP available". However, this choice is not completely free: the selection of a theorem prover will be largely dictated by the characteristics of the problem.

First, it is necessary to select a logic theory that allows describing and reasoning about our theorem. For example, if we need to prove a property of prime numbers, our logic theory should be able to express concepts like "number", "division" or "prime" and draw conclusions from these concepts. Some examples of logic theories are the following:

- There are several logic theories that allow reasoning about numerical properties. The difference between these theories is the domain of numbers being considered (naturals, integers, reals, ...) and the set of supported operations (equality, addition, multiplication, ...). Some logic theories within this category are Difference logic (supports the comparison of differences of two variables and constants, *e.g.* $x\text{-}y \geq 5$, decidable in polynomial time), Presburger arithmetic (supports addition and equality of natural numbers, decidable in doubly exponential time) and Peano arithmetic (supports addition, multiplication and equality of natural numbers, undecidable).

- There are logic theories for reasoning on other abstract mathematical concepts such as equivalence relations (*e.g.* equality), order relations (*e.g.* the "less than" operator), Boolean algebras, sets, graphs, ...

- *Modal logic* is a family of logics that allows the qualification of the truth of a statement, *e.g.* "it is *always* true", "it is *quite* true". An example is *deontic logic*, which can be used to express statements of obligation, *e.g.* "x is mandatory" or "y is optional".

- *Temporal logic* is a family of modal logics that can express properties about time, formalising properties like sequence ("A after B"), temporal causality ("A until event B") or duration ("A lasts for 10 minutes"). Some examples of specific logics in this family are Linear Time Logic (LTL) and Computational Tree Logic (CTL). This logic is used frequently to reason about the dynamic behaviour of programs, protocols, hardware designs,... Specialised tools used to reason with temporal logic formulae are called *model checkers*.

- *Fuzzy logic* is used to model uncertainty. Rather than being "true" or "false", the truth-value of a fuzzy logic statement is a real value between 0 and 1 (1 being completely certain and 0 meaning completely false). This type of logic is used, for example, in artificial intelligence to reason about complex problems in which there is ambiguity or incomplete information.

- *Description logic* is a family of logics dealing with the representation of knowledge (concepts, individuals and relationships among them). For example, we can define the statements "A dog is an animal" or "Snoopy is a dog" and reason about "is Snoopy an animal?". These logics are the reasoning engine behind ontologies and Semantic Web technologies. An example of a language based on this logic is OWL (Ontology Web Language).

For a specific problem, there may be several logic theories that are adequate. In that case, using the least expressive logic theory that is sufficient to solve the problem is the best strategy. The rationale is that there is a direct relationship between the expressive power of a logic theory and its theoretical and practical limitations: more expressive theories may be undecidable or their decision procedure may be computationally more complex.

Once we have determined a suitable logic theory, the choice of a theorem prover is straightforward: each theorem prover provides a comprehensive list of the logic theories that it supports. Hence, it is possible to build a catalogue of candidates. From this list, the selection will be based on several criteria:

- *Automation:* If possible, choose an ATP over any ITP. The only exception to this rule is when the ATP is not efficient enough for our purpose.

- *Efficiency:* Select the most efficient theorem prover from those available. The automated reasoning community performs yearly research competitions that evaluate the state-of-the-art tools, and these benchmarks are very useful for comparison.

Complementary reading

Hodges W. (1991) *Logic: an introduction to elementary logic*. Penguin books.
Robinson A., Voronkov A. (2001) *Handbook of Automated Reasoning Volume I & II*. Elsevier and MIT Press.
Baader F., Calvenese D., et al (2003) *The Description Logic Handbook*. Cambridge University Press.
Gabbay D.M. et al (2007) *Handbook of Modal Logic*. Elsevier.
Novák V., Perfilieva I. (2001) *The Principles of Fuzzy Logic: Its Mathematical and Computational aspects*. In Lectures of Soft Computing and Fuzzy Logic, pages 189–237. Springer-Verlag.
Wang F. (2004). *Formal Verification of Timed Systems: A Survey and Perspective*. Proceedings of the IEEE, 92(8):1283-1305.

- *Maturity:* Ideally, the tool should have a established community and active support.

- *Usability:* Ideally, the tool should have GUIs for its configuration and execution, even though this is not the norm for ATPs, which are mostly command-line-based. Also, the tool should have sufficient documentation: user manuals, user mailing lists, forums, portals, lists of frequently asked questions and GUIs. We should take into account that these tools are created by experts for experts and, consequently, they may not be extremely user-friendly to the newcomers.

Using a theorem prover

The premises and the theorem must be written in the input language of the prover. Very often, the tool manual provides a definition of the syntax of this formalism and the homepage of the tool provides examples that can be used as a starting point.

Even though understanding the language is usually simple, the formalisation of the problem is a complex task. It might be necessary to include many premises that in a pen-and-paper proof are taken for granted, *e.g.* that any number is equal to itself or that 0 is smaller than any other natural number. Some of these axioms may be already included in the logic theory. For instance, if the logic theory includes equality, the theorem prover will already "know" that any number is equal to itself. Nevertheless, if this information is required for our proof and it is not part of the logic theory, it should be formalised explicitly, otherwise the theorem prover will fail to complete the proof.

We should also make sure that our formalisation does not include contradictory premises. If the premises are contradictory, the theorem prover can conclude anything: for example, we will also be able to prove the negation of our theorem! Some theorem provers provide a warning if the premises are contradictory, but otherwise the absence of contradiction should be double-checked.

Regarding the use of an interactive theorem prover, it is not possible to provide general advice about its use. The tool documentation, examples and forums should provide detailed information about the tool.

4. Planning formal proofs

Formal proofs are rigorous but they are far from being mechanical. Usually, there are several potential proof strategies and not all of them may lead to the conclusion. Furthermore, other decisions like the choice of notation can greatly influence the difficulty of the proof and its readability. Making the right choices requires skill and this skill is learnt mostly from experience, *i.e.* reading proofs of others and proving properties yourself.

This section aims to describe the steps in the construction of a proof, identify the different decisions involved in a proof (notation, structure, ...), and the factors that must be considered in each decision. Also, we provide general guidelines for each of those decisions. Unfortunately, each proof is different and it is not possible to provide detailed guidelines that are valid for any property and problem domain.

The development of a proof can be divided into the following nine activities:

Number	Activity name	Products
1	Identify the problem	Informal problem statement
		Keywords
2	Review the literature	Relevance of the problem
		Problem status (open or proved?)
		Related problems and proofs
		Suitable notation and vocabulary
		References
3	Identify the premises	Premises
		Corner cases
4	Understand the problem	Intuition about the problem
5	Formalise the problem	Formal problem statement
		Choice of a logic formalism
		Notation
		Definitions
6	Select a proof strategy	Proof strategy
7	Develop the proof	Initial proof
8	Re-evaluate the proof	Refined proof
9	Improve readability	Final proof

Even though these activities are numbered sequentially, writing a formal proof is not a sequential process, but rather an iterative one: each activity may provide new insights that can be used to rework the result of previous activities. It is common, for example, to revise the notation once we get into a specific proof. Activities 8 (re-evaluate the proof) and 9 (improve readability) intend to make this iterative process explicit.

4.1. Identifying the problem

The process starts when we discover a statement that needs to be proved. For example, this property may be the result of design and creation research: Is this method correct? Is this algorithm optimal? Is this architecture tolerant to failures? The list on Section 1 may provide an initial starting point to define the problem statement.

The irrelevant details of the problem should be abstracted. For example, if we are proving a property of a system with two processes, does our solution qualify for systems with N processes as well? If we are studying a property of a network, can we abstract the fact that it is a network and model it as a graph? It may happen that at this step it is not possible to determine what is relevant and what is not. This is not a problem as the following activities will let us refine our abstraction.

The desired product of this step is an informal description of the problem, identifying the set of relevant keywords ("network", "algorithm", "efficiency", "decidable", ...).

4.2. Reviewing the literature

Next, we should try to answer several questions to make sure that the proof will be a useful contribution to knowledge: Is the problem well known in our research field? Has the problem drawn interest? Is the proof of this problem considered relevant or necessary? Is the problem related to other problems?

As a rule of thumb, the relevance of a problem is an indicator of the research value of its proof. A well-known problem, which is considered *open* because it has not been solved and which has important implications, is a good candidate for a formal proof.

A note on open problems

In general, open problems should be approached with caution. On the one hand, being open means that the result has not been proved yet and there is an opportunity to contribute. On the other hand, some open problems have been previously studied by the research community for many years or even centuries. Therefore, if they remain open it is because their proof is very complex or it requires new developments before it can be completed. It is recommended to gather substantial research experience before selecting the proof a well-known open problem as the research goal.

Suggested reading

Keith J. Devlin (2002). *The Millenium Problems: The seven greatest unsolved mathematical puzzles of our time*. New York: Basic Books.

Statements that contradict the folklore fall into the same category. This type of proofs is extremely rare, but these proofs do happen. However, in these situations the odds are against us: the most likely explanation is that our proof is wrong. Also, claims about the proof of open problems or bold statements may be subjected to intense scrutiny or, even worse, they may be rejected directly as bogus. Therefore, it is extremely recommended to double-check the results repeatedly and to enlist colleagues and peers in the revision of such important proofs.

Sometimes, the property we are trying to prove may be specific to a particular computer system, algorithm or architecture. In that case, it is necessary to check that the property we want to prove is considered relevant for that type of system. For example, in some contexts proving the termination of an algorithm might be important, while in other fields it is taken for granted.

If the result has been previously proved, that is not the end of the story. Better proofs of important results (more compact, more elegant, easier to understand) provide new insights about the problem and are therefore a valid contribution. However, the quality standards for evaluating a reproof are set much higher than those for a new proof.

Our second goal with this literature review is gathering hints on how to construct your proof. Are there any intermediate results that can be used in the proof? What is the accepted notation and terminology to talk about the problem? How were similar properties proved in the literature? What is the degree of formality and detail of those proofs?

4.3. Identifying the premises

Some statements are universal and hold in any possible scenario, while others hold only on a restricted set of scenarios, *e.g.* only if there are three or more processes, only if the input is non-empty, only if the input number is prime, only if the cryptographic key is never reused, ... In the latter case, it is necessary to identify the scenarios in which the property will be studied: these will be the initial premises.

When selecting the premises, a typical decision is whether or not the property applies to the *corner cases*. Identifying the set of corner case is completely problem-dependent. At least, we should check whether any of the following questions applies to the problem:

- Can any entity of the problem be empty? *E.g.* a string, an array, a list, a graph, a set, ...

- Can any entity of the problem be full up to its maximum capacity? *E.g.* an array, a cache, a memory, a register bank, a hardware pipeline, ...

- Can any entity of the problem be undefined? *E.g.* a null pointer, a null attribute of a table, ...

- Are there numerical variables whose values may have special significance? *E.g.* zero, one, infinity, minimum or maximum capacities, off-by-one values ($N \pm 1$), ...

- Could two entities within the problem be equal among them? *E.g.* two pointers referring to the same memory address, two processes accessing the same resource, a node being the source and target of a message, ...

- Could one entity within the problem be included into another? *E.g.* a page already in a cache, an element already in a data structure, a tuple already in a database, a network that is included into another, a subset of a larger set, ...

At this stage, it is acceptable to consider basic scenarios only and forbid some or all these corner cases. Later, we can always broaden the scope of the property if we realise that it is applicable to more scenarios. Anyway, it is fundamental to keep track of these corner cases during all steps of the proof.

4.4. Understanding the problem

Before writing, we should understand *why* the property we are trying to prove holds. A good starting point is trying to find a specific case in which the property does not hold, a *counterexample*. Bear in mind that the property we are trying to prove might be false. Thus, do not be surprised if we actually find one! If we find a counterexample, we should consider adding a new premise that forbids the counterexample before discarding the proof entirely.

The first counterexample to be tested should be the simplest scenario. If the property holds in this case, we can later proceed towards more complex scenarios, trying to craft a scenario in which the property does not hold. Corner cases should be evaluated as well. The goal of this process is understanding the rationale behind the impossibility of finding a counterexample: this insight is what needs to be captured in the formal proof.

4.5. Formalising the problem

It is now time to provide a formal statement of the problem to be proved. The problem statement should take into account multiple goals: it should be compact, precise, unambiguous and easy to understand.

Before describing the property, some preliminary definitions may have to be introduced to provide a proper background. For instance, let us consider a proof about the efficiency of an algorithm for solving systems of linear equations. In this case, we should start by defining what is a linear equation, what is a system of equations and what is considered a solution to this system of equations. These definitions will greatly simplify the discourse throughout the rest of the proof.

The choice of a logic formalism will determine which type of deductions can be performed during the development of the proof. In Section 3, we introduce several logic formalisms and their application domains. The review of the literature can reveal which logics are typically used in the current research field. If there is no explicit mention about which logic is being used, it is because propositional or first-order logic is assumed. There is a more detailed discussion of these logics in the proof development section (4.7.).

Regarding mathematical notation, it is important to ensure that it is used but not abused. Mathematical symbols represent complex expressions compactly and unambiguously. However, an overabundance of notation may hide the meaning of expressions, making them non-obvious. Ideally, we should strive to keep the right balance between precision and understandability. This decision will largely depend on the audience and the usual proof style in the literature of the field. As an example, consider the following equivalent definitions, each of which may be preferred in a different context:

- "Given a non-empty set of reals S, the supremum $\sup(S)$ is the smallest real that is greater than or equal to all elements of the set".
- "$S \subseteq \mathbb{R}, S \neq \emptyset : \quad \sup(S) \stackrel{def}{=} \min(\{\, x \in \mathbb{R} \mid \forall y \in S : x \geq y \,\})$".

Whenever it is possible, the formalisation should use well-established vocabulary and notation. If our definitions do not match those in the literature, we should seriously consider using different terms: using notation and terminology with non-standard meaning is a recipe for disaster. For concepts that do not appear in the literature because they are introduced in the current proof, new definitions and notation can and should be introduced as needed. Finally, definitions and notations that are used verbatim should be referenced adequately in the proof.

4.6. Selecting a proof strategy

There are several approaches to tackle a proof, some of which may be more adequate to prove a specific type of properties. The selection of the best proof strategy for a given proof is kind of an art: there are some general criteria, but each problem is different. Section 5 provides a brief description of several proof strategies, together with information about the type of properties in which they can be most useful.

Suggested reading

Antonella Cupillari (2005). *The Nuts and Bolts of Proofs*. Burlington: Elsevier Academic Press, 3rd edition.

4.7. Developing the proof

Once we have established the goal of the proof and determined which axioms, definitions and notations are available, it is time to start the deduction process. In every step, we should reach a conclusion from the previous statements (premises), axioms or definitions.

What is a "valid" conclusion from a set of premises is determined by the inference rules of logic. There are different families of logic (propositional, first-order, high-order, modal, fuzzy, ...), each with a different catalogue of inference rules. In most situations, we will not explicitly consider which logic is being used, although we are implicitly working on propositional or first-order logic.

A detailed overview of inference rules in all families of logic is out of the scope of this module. As a reminder, we focus on propositional logic, in which logical formulae are constructed from the combination of the following building blocks:

- Propositional variables (A,B,...), in which each variable is either true or false.
- The constant "true", a proposition that is always satisfied.
- The constant "false", a proposition that is never satisfied.
- The operator for the negation of a formula ($\neg$), which is true if the formula is false and vice versa.
- The operator for the conjunction of two formulae or "logical and" ($\wedge$), which is true only if both formulae are true.
- The operator for the disjunction of two formulae or "logical or" ($\vee$), which is true if one or both formulae are true.

Other common logical operators such as implication ($\rightarrow$) or the double implication ($\leftrightarrow$) can be defined from this base operators. The following table describes a set of axioms, definitions and inference rules from propositional logic which can be useful in many formal proofs:

$\neg$true	$\equiv$	false	The negation of a true statement is false
$\neg$false	$\equiv$	true	The negation of a false statement is true
$\neg(\neg A)$	$\equiv$	A	The negation of a negation is the original
true $\wedge A$	$\equiv$	A	Definition of conjunction ($\wedge$)
false $\wedge A$	$\equiv$	false	Definition of conjunction ($\wedge$)
$A \wedge A$	$\equiv$	A	Definition of conjunction ($\wedge$)
$A \wedge B$	$\equiv$	$B \wedge A$	Commutativity of ($\wedge$)
$A \wedge (B \wedge C)$	$\equiv$	$(A \wedge B) \wedge C$	Associativity of ($\wedge$)
$A \wedge (\neg A)$	$\equiv$	false	A statement cannot be both true and false
$A \wedge (B \vee C)$	$\equiv$	$(A \wedge B) \vee (A \wedge C)$	Distributivity of $\wedge$
$\neg(A \wedge B)$	$\equiv$	$(\neg A) \vee (\neg B)$	De Morgan's Law (Negation of $\wedge$)
true $\vee A$	$\equiv$	true	Definition of disjunction ($\vee$)
false $\vee A$	$\equiv$	A	Definition of disjunction ($\vee$)
$A \vee A$	$\equiv$	A	Definition of disjunction ($\vee$)
$A \vee B$	$\equiv$	$B \vee A$	Commutativity of ($\vee$)
$A \vee (B \vee C)$	$\equiv$	$(A \vee B) \vee C$	Associativity of ($\vee$)
$A \vee (\neg A)$	$\equiv$	true	Statements are either true or false
$A \vee (B \wedge C)$	$\equiv$	$(A \vee B) \wedge (A \vee C)$	Distributivity of $\vee$
$\neg(A \vee B)$	$\equiv$	$(\neg A) \wedge (\neg B)$	De Morgan's Law (Negation of $\vee$)
$A \rightarrow B$	$\equiv$	$(\neg A) \vee B$	Definition of implication ($\rightarrow$)
$A \rightarrow B$	$\equiv$	$(\neg B) \rightarrow (\neg A)$	Contrapositive
$\neg(A \rightarrow B)$	$\equiv$	$A \wedge (\neg B)$	Negation of an implication
$A \leftrightarrow B$	$\equiv$	$(A \rightarrow B) \wedge (B \rightarrow A)$	Definition of if-and-only-if
$\neg(A \leftrightarrow B)$	$\equiv$	$(A \wedge \neg B) \vee (\neg A \wedge B)$	Negation of if-and-only-if
$A \wedge B$	$\vdash$	A	If (A and B) holds, then A also holds
A	$\vdash$	$A \vee B$	If A holds, then (A or B) holds
A,B	$\vdash$	$A \wedge B$	If A and B both hold, then (A and B) holds
$A \rightarrow B$,A	$\vdash$	B	If (A implies B) and A hold, then B holds

The symbol ($\equiv$) denotes that two logical expressions are equivalent: whenever we find the left-hand side expression, we can replace it by the right-hand side and vice versa. Meanwhile, the symbol ($\vdash$) denotes that the right-hand side ex-

pression is a logical conclusion of the left-hand side: whenever we find the left-hand side, we can conclude the right-hand side, but it does not necessarily work in the opposite direction.

Special care should be taken to avoid falling into a *logical fallacy*, that is, an incorrect application of an inference rule whose conclusion is not supported by the premises. A classical logical fallacy is the reversal of an inference rule. For example, $A \wedge B$ cannot be deduced simply from A and A cannot be deduced from $A \vee B$. Another common type of logical fallacy is confusing cause and consequence in an implication. If A implies B ($A \rightarrow B$), the fact that B holds does not mean that A holds: B could be satisfied independently as well, otherwise the formula should be a double implication ($A \leftrightarrow B$).

Examples of fallacies

Correct deduction	Incorrect deduction
n is even and greater than 5. Therefore, n is even.	n is even. Thus, n is even and greater than 5.
Peter is 18 years old. Therefore, Peter is an adult (he is 18 or older).	Peter is an adult (he is 18 or older). Therefore, Peter is 18 years old.
If it rains, the floor becomes wet. It rains, therefore the floor is wet.	If it rains, the floor becomes wet. The floor is wet, therefore, it must be raining.

To let readers verify the correctness of the formal proof, they should be able to understand the chain of deductions. We should always try to provide sufficient information about every step: Is it an axiom (which one)? Is it deducted from one of the definitions (which one)? Is it a consequence of one or more previous statements (which ones)? Are we referencing a previous theorem or lemma (which one)?

The level of detail to be used depends largely on the audience. Considering the venue in which the proof will be presented and the information from literature review, it is possible to identify the background of the average reader and select the appropriate level of detail. After that, trivial steps may be omitted and some knowledge of fundamental axioms of the field may be assumed. This allows us to simplify the proof.

4.8. Reevaluating the proof

Once the proof is complete, it is time to review it critically. Our primary concern will be locating errors in the proof. After that, we will examine the brevity, understandability and elegance of the proof.

Checking the correctness of a formal proof is a complex and time-consuming process, as errors may be very subtle. First, we should revise each inference step of the deduction, checking that it is either an axiom or a conclusion of the premises. For further assurance, usually the only option is relying on an

expert reviewer, who must be knowledgeable both on the specific topic and formal proofs. Another option, which is discussed in Section 3, is the use of a theorem prover.

Terms like "obvious" or "trivial" and sentences like "this is left as an exercise to the reader" or "details are omitted for brevity" should be used scarcely and carefully. First, applying terms like "trivial" to complex statements may be considered a sign of laziness or lack of rigor ("trivial" means "did not bother to prove that"). Moreover, some readers may find that those statements are not trivial and therefore may be discouraged by repeated uses of the term. Finally, the errors in a proof are often hidden in deceptively simple statements that are not checked because they seem obvious or their proof is very long. It is important to make sure that when a statement is called "trivial" it is because (a) its truth will be obvious to any reader of the proof and (b) there is evidence that the statement is true, like a detailed proof.

Fermat's last theorem

The last theorem of Pierre de Fermat (1601-1665) became infamous because of its lack of proof. The author claimed: *"I've found a remarkable proof of this fact, but there is not enough space in the margin to write it"*. This theorem was proved by the mathematician Andrew Wiles (1953-.) in 1995.

After ruling out errors, it is time to consider alternatives that may improve the quality of the proof: Can we choose a more compact and elegant notation? Is there another proof strategy that is more adequate for the problem?

A final point of evaluation is the structure of the proof: as in computer programs, modularity is a plus. A large proof may be hard to follow: can it be split into smaller subproofs? Instead of proving a complex theorem directly, it is better to start by proving auxiliary properties (*lemmas*) that can be later used within the proof. Lemmas should be chosen carefully to improve the readability of the complete proof. For instance, a property that appears several times within a proof is a good candidate to become an independent lemm.

4.9. Improving readability

Throughout the development of the proof, we have already taken the audience into account: when choosing the notation, the vocabulary, . . . However, we can still improve the writing in several ways in order to provide the final polish.

First of all, we should make sure that we have correctly signalled both the start and end of the proof. Highlighting the proof strategy is also recommended.

In our natural language explanations, we should convey the logical flow of the deduction process. Sentence connectives are instrumental towards conveying the relationship between the current statement and the premises. It is a good idea to use a variety of connectives to avoid repetition.

The first property that must be described is cause and consequence. It is very important to clearly distinguish between them, as reversing cause and conse-

quence is a common error and the root of many logical fallacies. The following table describes several appropriate connectives:

Pattern	Sample connectives
cause **connective** *consequence*	implies, means, so, if-then
consequence **connective** *cause*	as, as long as, because (of), due to, for, inasmuch, owing to, since
connective *cause*, consequence	as, as long as, because (of), due to, owing to, since
connective, consequence *cause*; **connective**, *consequence*	as a consequence, as a result, consequently, for this reason, hence, in that case, therefore, thus
connective consequence	that is why, this implies (that)

Examples

If there is a cache miss **then** the page will be fetched from memory.
We know that there is a cycle in the graph **because** it is a complete graph with more than 3 vertices.
Since x is even, $x + 1$ is odd.
All cases have been considered; **therefore**, the proof is complete.
This implies that the output is always positive.

We should avoid using cause and consequence connectives in statements that are not the consequence of the premises. If we want to provide a clarification, we can use "in other words", "that is" or "i.e.". To introduce an example, we can use the connectives like "for example", "for instance" or "*e.g.*".

In those statements that concern a new aspect of the proof, it is better to use connectives like "moreover", "also", "on the other hand", "besides",... Another strategy to signal a shift in the discussion is starting a new paragraph.

After reviewing the proof, we can identify the most complex steps. It is a good idea to add clarifications or reminders in natural language, in order to guide readers. Also, we should consider including examples or figures. Again, letting a colleague or peer examine your proof critically is a good practice.

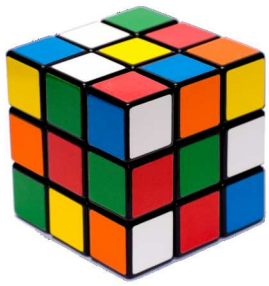
5. Proof strategies

Writing a proof has many things in common with writing a program. A program is a detailed sequence of instructions that compute a result, while a proof is a sequence of logical inference steps that deduce a conclusion. In the same way as we can often write several programs that compute the same result (some smaller and more elegant than others), there are usually several valid ways to prove a property. Furthermore, both in writing programs and proofs, the overall approach used to attack the problem has a direct impact in the quality of the solution.

In the area of formal proofs, the approach used to plan a proof is called *proof strategy*. In order to engineer a good formal proof, choosing the most suitable proof strategy is very important. There are some basic rules about when each strategy should be preferred, but there is also some degree of creativity involved.

Proof strategies are not tied to a particular logic formalism, but rather general patterns that are valid in different logics (we may call them "metalogic patterns"). Continuing our parallelism with programming, proof strategies would be the equivalent of algorithm design paradigms such as divide and conquer or backtracking: general guidelines to solve a problem which are both problem-independent and language-independent.

In this section, we present a collection of basic proof strategies, together with indications on when they should be used and which are the common errors that may occur when we apply them. Also, for each method, we provide a sample proof to illustrate the method and provide additional insight on when it should be used.



A Rubik's cube.
Source: Wikipedia, User "Alvaro qc"
License: Creative Commons Attribution 3.0 Unported.

Direct proof

This proof strategy can be used when the premises, related axioms and definitions provide enough information to prove the result directly. Each step, the statement is either an axiom or a conclusion of previous steps.

Theorem A Rubik's cube has a finite number of configurations.

Proof (direct): A configuration of a Rubik's cube is completely determined by the colour of each sticker. A Rubik's cube has 6 faces and each face has 9 colour stickers, *i.e.* 54 stickers in total. Each colour sticker has 6 possible colours. Therefore, the number of configurations is less than 54^6. In fact, it is much lower because there can only be at most 9 stickers of the same colour, but this upper bound suffices to complete the proof. ■

Proof by example or counterexample

An *example* of a property is a specific instance or scenario in which the property holds and a *counterexample* is an instance in which the property does not hold. Depending on the property under study, an example or counterexample may be sufficient to prove or disprove the property.

Let us consider a specific property:

Definition (Modulo): The *remainder* or *modulo* of dividing x by y is denoted as $x \equiv k \bmod y$. For example $7 \equiv 3 \bmod 4$, $3 \equiv 1 \bmod 2$ and $5 \equiv 5 \bmod 6$. If $x \equiv 0 \bmod y$ we say that "x is divisible by y" or that "x is a multiple of y".

Definition (Prime): A prime is an integer number greater than 1 which is only a multiple of 1 and itself. For example, 2, 3, 7 and 11 are primes. Examples of non-primes include 1 (by definition), 4, 21 or 70. Formally, a number x is prime, denoted as $prime(x)$ if the following holds:

$$prime(x : \mathbb{Z}) \stackrel{def}{=} (x > 1) \wedge (\forall y \in \mathbb{Z} : (x \equiv 0 \bmod y) \rightarrow (y = x \vee y = 1))$$

Goldbach's Conjecture: Any even integer greater than 2 can be expressed as the sum of two prime numbers. Formally:

$$\forall a \in \mathbb{Z} : (a > 2 \wedge a \equiv 0 \bmod 2) \rightarrow (\exists b,c \in \mathbb{Z} : prime(b) \wedge prime(c) \wedge a = b + c)$$

Goldbach's conjecture

This conjecture was proposed by the mathematician Christian Goldbach (1690-1764) in a letter to the mathematician and physicist Leonhard Euler (1707-1783). It has been confirmed by a computer for all even numbers up to $15 \cdot 10^{17}$, but there is still no proof of its validity in general.

Let us consider an even number greater than 2: 16. This number can be decomposed as the sum of two primes 5 and 11 and, thus, it is an "example" of a number satisfying Goldbach's conjecture. Other examples are 30 (the sum of 13 and 17) or 62 (the sum of 31 and 31). The fact that some examples satisfy this property does not constitute a proof, as the theorem affects all even numbers greater than 2.

On the other hand, as the property applies to all even numbers greater than 2, identifying a counterexample (an even number greater than two in which the property does not hold) would be sufficient to disprove Goldbach's conjecture. Interestingly, up to date there are no known counterexamples of this property: all even numbers in which this property has been checked satisfy Goldbach's conjecture. Notice that the lack of counterexamples does not constitute a proof of the conjecture "per se": it is possible that some number that has not been tested yet does not fulfil the property. A proper proof would need a formal justification of the impossibility of the existence of a counterexample.

Complementary readings

A novel of fiction about the efforts to prove Goldbach's conjecture:
Dioxadis A. (2001) *Uncle Petros and Goldbach's conjecture*. Faber and Faber.

Proof by cases / exhaustion / brute-force

Consider a finite set of *cases* that cover all possible scenarios of a problem. If the property can be proved separately for each case, the property holds in general. The definition of cases should be *exhaustive*, *i.e.* it should ensure

that it captures all scenarios. The most common mistake in this type of proof is proposing a list of cases that does not consider some specific scenario in which the property does not hold.

Theorem During a game of chess, there are at most 32 pieces on the board at all times.

Proof (by cases): According to the rules of chess, in the initial configuration there are 16 pieces of each colour (8 pawns, two rooks, two knights, two bishops, one queen and one king). The moves in a game of chess can be classified into three categories:

- Moves in which no piece is removed from the board, which include basic moves and castling. This type of move does not change the number of pieces on the board.
- Pawn promotions, in which a pawn reaches the final column of the opposite colour and can transform into the piece of the player's choice. Notice that pawn promotions do not change the number of pieces either, as the pawn is replaced by another piece.
- Moves in which one piece captures another (including *en passant* captures). In a capture, the number of pieces of the board is reduced by one.

We have seen that the initial configuration of the board has 32 pieces and that no move can add pieces to the board (at most, it can replace an existing piece by another). Therefore, we can conclude that the number of pieces in the board is always less or equal to 32. ■

If the number of cases is small, the proof is typically performed by hand. However, if the number of configurations is very large, it is good idea to write a program that checks all cases mechanically. Two famous examples of computer-assisted brute-force proofs are the *four colour theorem* (it is possible to paint any map in such a way that adjacent regions have different colours using at most four colours) and *God's algorithm* for solving a Rubik's cube (the minimum number of moves required to solve a Rubik's cube from any initial configuration, which is discovered by considering all possible configurations!).

Proof by contradiction / reduction to the absurd

Start by assuming the negation of what you want to proof and try to reach a contradiction. The contradiction shows that the assumption is impossible as it leads to a logical fallacy. Therefore, we can conclude that the negation of the assumption (that is, our original theorem) holds.

An advantage of this approach is that it provides an initial statement from which deductions can be drawn. Like the contraposition method that is introduced next, this approach is useful when it is hard to reason about a property but its negation is simpler to inspect, *e.g.* properties about infinity or properties about universal quantifiers (properties that hold for all elements of a collection).

Theorem There are infinitely many prime numbers.

Proof (by contradiction): Intuitively, we prove that if we try to set an upper bound to the number of primes, we can always find another prime beyond the hypothetical largest prime.

An ancient proof

This proof of the infinitude of prime numbers dates from the text *Elements* (Book IX, Proposition 20) from the Greek mathematician Euclid (around 300 BC).

Assume that the set of primes is finite, formed by the primes $p_1, \ldots, p_n$. Then, let us study the set of divisors of $N = (p_1 \cdot p_2 \cdot \ldots \cdot p_n)+1$. By this definition, any number outside these p_i cannot be a prime number, *e.g.* any number larger than p_n is not a prime. Also, N is not a multiple of p_1 because it is equal to a multiple of p_1 + 1 (dividing N by p_1 would produce a remainder of 1). Using a similar argument, we can show that N is not a multiple of any of the remaining primes.

The remaining question is: is there a number that can divide N?

- If the only divisors of N are 1 and itself, then N is a prime (by the definition of prime) and we have reached a contradiction: we had previously deduced that N should not be a prime, as it is larger than p_n.

- If there are divisors of N other than 1 and N, let us denote as x the smallest of such divisors that is greater than 1. We have that x cannot be divided by any number from 2 to x - 1 as we have explicitly selected the smallest divisor of N that is greater than 1. Therefore, x must be a prime. However, x cannot be a number from the list of primes $p_1, \ldots, p_n$ because we have established that no p_i is a divisor of N. There is a contradiction, as x must be prime and cannot be prime simultaneously.

In both scenarios we reach a contradiction. Therefore, our initial premise must be contradictory: there are infinitely many primes. ■

A typical error in proofs by contradiction is misunderstanding the negation of a statement. Even though it seems straightforward, we should double-check our conclusions to make sure that they correspond to the negation of our initial assumption. For example, suppose that we assumed that "there is a number in set A which is greater than 20" and reached a contradiction. Does this mean that there is a number in A which is smaller than 20? No, the correct conclusion would be "either is A empty or it contains numbers that are all smaller or equal to 20". Notice that we cannot guarantee that there is a number and we cannot guarantee either that it is strictly smaller than 20 (it could be equal to 20).

Proof by contraposition

If the property to be proved is an implication (A implies B or B is a consequence of A, denoted mathematically as $A \rightarrow B$), it is equivalent to prove the *contrapositive*: that the negation of B implies the negation of A, *i.e.* $\neg B \rightarrow \neg A$.

Theorem Let x and y be two integers. If x^2 and y^2 are different, then x and y must also be different. Formally:

$$\forall x,y \in \mathbb{Z} : (x^2 \neq y^2) \rightarrow (x \neq y)$$

Proof (by contraposition): The contrapositive of our theorem is the following equivalent statement: "if x is equal to y, then x^2 is equal to y^2":

$$\forall x,y \in \mathbb{Z} : (x = y) \rightarrow (x^2 = y^2)$$

Notice that the negation of $x \neq y$ (x and y are different) is $x = y$ (x and y are equal). It follows directly that if $x = y$, then $x^2 = x \cdot x = y \cdot y = y^2$. ■

Proof by mathematical induction

Induction is commonly used when proving a universal property over an infinite collection, for example, a property satisfied by all numbers, or by all strings or by all sets of integers. It is not possible to prove this type of property by exhaustion because the number of cases is infinite.

The proof starts by checking that the property holds for the "first" element of the collection, what is called the *base case*. Then, we show that if the property holds for some arbitrary element in the collection, it also holds for the "next" one(s) in the collection. The combination of (a) being satisfied for the first element and (b) being satisfied by an element if it is satisfied by the previous one creates a *domino effect*: as the first element satisfies the property due to (a), the second one also satisfies the property due to (a + b); Then, the third element satisfies the property due to (a + b + b), the fourth also satisfies the property due to (a + b + b + b) and similarly for all elements of the collection.

The definition of "first" and "next" depends on the nature of the elements in the collection. For example, for integer numbers the "first" will be the one with the lowest value and the next value for n will be $n+1$. For strings, we can define the "first" as the one with the least symbols, *e.g.* the empty string, while the "next" strings of s would be all strings with one more symbol than s. For sets, the "first" would be the set with the least number of elements (the empty set $\emptyset$) while "next" would be sets with one more element than the previous one.

A term that is commonly used in induction proofs is *induction hypothesis*. When we are trying to prove that "property(n)" implies "property($n + 1$)", we assume that "property(n)" holds and then use this statement to deduce "property($n + 1$)". The assumption "property(n)" is called induction hypothesis and it is important to define it explicitly in the proof ("This is our induction hypothesis") and to pinpoint when it is being used ("By induction hypothesis, . . . ").

Theorem For every integer n, the sum of all the numbers from 0 to n is equal to n times $n + 1$ divided by 2, which can be expressed mathematically as:

$$\sum_{0}^{n} i = \frac{n \cdot (n+1)}{2}$$

Proof (induction): The property holds if $n = 0$, as the sum $\sum_0^0 i$ is 0 and the product $(0 \cdot (0 + 1)/2)$ is also 0. We assume that the property holds for an arbitrary value of n, which we refer as k, *i.e.* $\sum_0^k i = \frac{k \cdot (k+1)}{2}$. This is our induction hypothesis. Then, we will have to prove that the equality also holds for $k + 1$, *i.e.* $\sum_0^{k+1} i = \frac{(k+1) \cdot (k+2)}{2}$.

$$\sum_{0}^{k+1} i = 0 + 1 + 2 + \ldots + k + (k+1) \qquad \text{Expansion of } \Sigma$$

$$= \boxed{(0 + 1 + 2 + \ldots + k)} + (k+1) \qquad \text{Associativity of addition}$$

$$= \sum_{0}^{k} i + (k+1) \qquad \text{Definition of } \Sigma$$

$$= \boxed{\frac{k \cdot (k+1)}{2}} + (k+1) \qquad \text{Induction hypothesis}$$

$$= \frac{k \cdot (k+1) + 2 \cdot (k+1)}{2} \qquad \text{Moving } (k+1) \text{ into the fraction}$$

$$= \frac{(k+1) \cdot (k+2)}{2} \qquad \text{Extracting the common factor} (k+1)$$

We have proved the equality in the base case $n = 0$, and for $n = k + 1$ if it holds for $n = k$. This concludes our proof by induction. ■

Counting arguments

Sometimes it is possible to gather information about a property by considering the number of elements that satisfy it. These proofs are called *counting arguments* and they can lead to simple and elegant proofs. For example, if we know that "there are 1,000 humans whose surname is Stevens" and "there are 50,000 students at UOC", we can automatically infer that "there is at least one student at UOC whose surname is not Stevens" (in fact, at least 49,000).

Counting arguments usually require enumerating arrangements of a collection of objects, *e.g.* permutations, combinations or partitions. Combinatorics is a branch of mathematics devoted, among other concerns, to studying these arrangements and providing useful procedures for counting them. Formulae for counting arrangements depend on several factors: whether all objects from the collection are being considered or not, whether the order among objects in the arrangement matters, whether there are duplicates in the original collection, ... An example of such formulae is the permutation without duplicates: a collection with n distinct objects can be ordered in $n! = n \cdot (n\text{-}1) \cdot \ldots \cdot 2 \cdot 1$ different ways.

Counting arguments in combinatorics, called *combinatorial proofs*, are of two types: *double counts* and *bijections*.

A double count defines a set in two different ways and attempts to define two different methods to count the elements in the set. As both definitions correspond to the same set, both counts must be equal.

Bijection arguments try to establish whether two sets have the same number of objects. In order to do this, we try to define a one-to-one mapping between the objects of the two collections. Both collections have the same number of elements if and only if the mapping exists and we can ensure that is indeed a one-to-one mapping, *i.e.* that in both sets each object is mapped to exactly one object of the other set. Precisely, there is a proof strategy called *diagonalisation* or *Cantor's diagonal* which can be used to disprove the existence of a bijection between two sets.

Diagonalisation

This technique, introduced by the mathematician Georg Cantor (1845-1918), can be used to prove that there is no one-to-one mapping between the set of integers and the set of reals. It was also instrumental in Alan Turing's proof of the existence of undecidable problems: formalising problems and programs as sets, it is possible to proof that there is no one-to-one mapping among them, *i.e.* there are problems for which there is no program that can solve them.

Another example of counting arguments is the *pigeonhole principle*: if we are trying to place m pigeons inside n pigeonholes and $m > n$, then at least one pigeonhole has two or more pigeons. This apparently obvious proposition can be used to prove many interesting properties if we are able to rephrase the problem in terms of pigeons and pigeonholes.

Definition (friendship): Friendship is a relationship between two people. If a person x is a friend of a person y, then y must be a friend of x.

Definition (degree): The degree of a person x is the number of friends of that person.

Theorem In any group of people, there is an even number of people with an odd number of friends, *i.e.* an odd degree.

Handshake lemma

This proof is related to a property from graph theory called *handshake lemma*: the sum of all degrees of vertices in an undirected graph is equal to two times the number of edges among them.

Proof (by double count): Let us consider the total number of friends in our group, *i.e.* the sum of the number of friends of each person.

There are two different ways to count this magnitude: adding the degrees of each person or by considering the total number of friendship relations in our group. If there are F friendship relations, adding the degrees will yield $2 \cdot F$ given that each friendship relationship is being counted twice, once per each person participating in the friendship. As both counts refer to the same concept, they must be equal:

$$\sum_{x \in \text{Group}} \text{degree}(x) = 2 \cdot F$$

As the result of adding all degrees is equal to $2 \cdot F$, it is an even number. Therefore the number of persons with an odd degree must be even, as otherwise the sum would be odd. ■

Summary

The following table summarises the proof strategies presented in this section.

Strategy	When to use it	Take care of
Direct proof	The path to P is clear from the start.	Avoid logical fallacies within the proof.
Exhaustion	P can be checked independently in a finite list of cases.	The list of cases should be exhaustive, covering every possible scenario.
Example	P is an existential property.	The example should be valid and it should satisfy P.
Counterexample	$\neg P$ is a universal property.	The counterexample should be valid and it should satisfy $\neg P$.
Contradiction	Reasoning on $\neg P$ is simpler than reasoning on P, *e.g.* P is a universal.	Avoid misinterpretations of $\neg P$, check that the hypothesis is non-contradictory.
Contrapositive	P is an implication $(A \rightarrow B)$, proving $(\neg B) \rightarrow (\neg A)$ is easier than proving P.	Remember that $(A \rightarrow B)$ is not the same as $(B \rightarrow A)$ or $(A \leftrightarrow B)$.
Induction	P is a universal property about an infinite set.	Do not forget to check the base case.
Counting arguments	P is a property of a collection whose size can be counted or compared to another collection.	Consider whether order matters and whether there can be duplicates or symmetries.

6. The Internet and formal proofs

The Internet may provide useful information for the researcher interested in writing a formal proof about a problem:

- Bibliography on formal proofs and proof strategies.
- Domain-specific bibliography: problem statement, related problems, examples of preferred proof styles in a specific research community, ...
- Information on open problems: catalogues of open problems, problem statement, status of the problem, intermediate results, proposed (or discarded) strategies for proof, ...
- Information on software tools for writing or checking proofs: tools for download, portals, mailing lists, ...

Recent developments caused by the Internet are *proof repositories* and *proof wikis*. These collaborative repositories attempt to collect formal proofs from different areas of mathematics, statistics and computer science. Some of these repositories are backed by automated or interactive theorem provers that can check the corresponding proofs. Two examples of such repositories are:

- ProofWiki [`http://www.proofwiki.org`]
- Metamath [`http://www.metamath.org`]

7. Examples of proofs in IS and computing

There are many examples of proofs in the literature. Some results date back to the theoretical foundations of computer science, while others illustrate recent results. In the following, we summarise several of these contributions.

The four colour theorem

The four colour theorem states that it is possible to colour the regions in a map using only four colours in such a way that two adjacent regions do not share a colour. The problem was stated in 1852, but it was not proved until 1976 by Appel and Haken. The proof was performed by exhaustion with the help of a computer and it considered more than 1900 distinct cases. The fact that the proof was computer-based was quite controversial for its time, arising some claims about the validity of such proofs.

Complementary reading

Gonthier G. (2008) *Formal Proof: The Four-Color Theorem*. Notices of the AMS, (55)11: 1383–1393.
Appel K., Haken W. (1976). *Every map is four colourable*. Bulletin of the AMS, 82:711–712.

Complexity of the primality test

Agrawal, Kayal and Saxena (2004) proposed an efficient algorithm for testing whether an integer is prime. This problem is directly related to the problem of discovering the prime factors of an integer, a mathematical problem with direct implications in the field of cryptography: fast factoring algorithms can be used to break the security of RSA, a widely used public-key cryptosystem. The fastest approach for factoring integers still takes time that is exponential with respect to the size of the input integer. This paper proposes an algorithm for testing primality in *polynomial* time. Knowing that a number is not prime does not yield its list of prime factors. Consequently, this result does not imply that factoring can be resolved in polynomial time. However, this proof was such a breakthrough that it made the cover of the *New York Times*.

Complementary reading

Agrawal M., Kayal N., Saxena N. (2004). *PRIMES is in P*. Annals of Mathematics, 160(2): 781–793.
Bornemann F. (2003). *PRIMES Is in P: A Breakthrough for "Everyman"*. Notices of the AMS, 50(5): 545–552.

The paper provides a description of the primality test and two proofs regarding its efficiency and correctness. In terms of efficiency, the authors demonstrate that the number of computation steps of the algorithm is polynomially bounded with respect to the size (number of bits) of the input integer. Regarding correctness, the proof justifies that the primality test returns "true" if and only if the number is a prime. The proof is decomposed into proving the two implications separately: that if the algorithm returns true, then the number is a prime; and that if the number is a prime, the algorithm will return true.

Mutual exclusion

Leslie Lamport provided the theoretical foundations of many algorithms used in concurrent and distributed systems. A landmark contribution is Lamport's *bakery* mutual exclusion algorithm (1985), which describes a procedure that allows the synchronisation of multiple processes attempting to access a shared resource concurrently. The algorithm ensures that at most one process will gain access to the resource at the same time.

Complementary reading

Lamport L. (1974) *A New Solution of Dijkstra's Concurrent Programming Problem*. Communications of the ACM, 17(8):453–455.

Hardware verification

Kaivola et al (2009) discuss the use of formal verification techniques to analyse the behaviour of a specific hardware processor. The paper provides a high-level overview of the verification flow, the formalism used to describe the hardware design, the tools involved in the process and the optimisations, heuristics and fine tuning required to complete all the proofs. Formal verification of hardware processors is a hot topic since a bug was discovered in a floating-point division unit of a popular processor. The bug caused the recall of millions of processors, with important economical losses, and provided a motivating example for the use of formal verification in hardware design.

Complementary reading

Kaivola R., Gughalg R. et al (2009) *Replacing Testing with Formal Verification in Intel®Core™i7 Processor Execution Engine Validation*. CAV 2009, LNCS 5643, pp. 414-429, Springer.
Pratt V. (1995) *Anatomy of the Pentium bug*. TAPSOFT 1995, LNCS 915, pp. 97-107, Springer.

8. Evaluating formal proof research

The following criteria provide a checklist to evaluate a formal proof:

1) Method: Is it a pen-and-paper proof, computer supported or hybrid?

2) Relevance: Is the property to be proved relevant? (*i.e.* is this assurance useful for some purpose?) Does the proof have consequences on other properties, problems or domains?

3) Novelty: Has the property been studied in the past? Was the property an open problem? Has it been proved previously using a different approach?

4) Precision: Are the problem statement and the premises defined clearly and unambiguously?

5) Strategy: Is the proof strategy to be used clearly identified? Is it adequate for the type of property being proved?

6) Difficulty: Is the proof trivial? Does it involve an inventive step at some point? Is the proof strategy atypical for the type of problem being proved?

7) Correctness: Is any premise contradictory? Is every step of the proof a logical conclusion from previous steps or an axiom? Are inference rules correctly applied in every step? Are auxiliary lemmas and definitions correctly used? Does the author consider the corner cases in all steps of the proof? Is the conclusion of the proof actually the initial goal?

8) Notation: Is the degree of formality suitable for the target audience? Is the choice of notation adequate for the property being proved and the target audience?

9) Modularity: Is the proof too large? Is it divided into auxiliary lemmas to facilitate its comprehension?

10) Detail: Is the level of detail suitable for the target audience? Are the irrelevant features hidden from the reader? Does the proof remind the necessary background information that is unavailable to the target audience? (axioms, definitions, . . .)

11) Comprehension: Is it possible to understand the proof? Does the proof provide aids to guide readers? (examples, intuitive explanations of the more complex steps, . . .)

12) Readability: Is there a logical connection between one argument and the next? Is the proof easy to read? Is the end of the proof clearly signalled to readers?

Advantages of formal proof research include:

- A formal proof provides a *complete* assurance about the validity of a property, in all potential scenarios and with no "if"s or "but"s.

- Proofs of general problems are non-perishable and continue to hold regardless of the technological progress.

- Formal proofs can complement and validate research results produced with other research strategies such as design and creation. Furthermore, in some cases a formal proof is a *requirement*, *e.g.* a mutual exclusion algorithm is worthless without a proof that it achieves mutual exclusion.

- Writing formal proofs is a skill that can be learnt and improved with practice.

Disadvantages of formal proof research include:

- Compared to design and creation research, formal proof does not offer something tangible to show. Therefore, it may be hard to convey the significance of a proof to researchers from a different field.

- Writing formal proofs requires good knowledge in the research area, knowledge on logics and mathematical notation, and making it readable depends on good writing skills.

- Checking the correctness of a formal proof is a time-consuming and complex process, which may require the assistance of an external expert reviewer.

- A formal proof of an open problem is an outstanding result and, as such, it may be received with scepticism. For some important open problems, the large number of (incorrect) proof proposals makes it difficult to get the attention of a reviewer.

- Formal proof is often not appropriate for venues focused on an empirical evaluation of research results.

9. Further reading

There are several books that discuss methods and techniques for writing formal proofs, such as those by Cupillari (2005), Velleman (1994), Solow (2004) and Sundstrom (2006). Furthermore, the "Notices of the American Mathematical Society" journal devoted an special issue to formal proof on December 2008. All these sources provide broad overviews on writing formal proofs and general advice valid for any research field.

Furthermore, each section has provided bibliography for specific topics related to formal proof, *e.g.* logic formalisms, theorem provers, examples of proofs or famous proofs.

For advice on proving specific problems, the interested reader should focus on the domain-specific literature of that research field. A good starting point may be surveys focusing on setting the theoretical foundations and identifying the key problems in the area.

Summary

This module has presented formal proof as a research strategy in computing and related fields. With a strong foundation on mathematics and logic, formal proofs can provide certainty about a property or validate the product of design and creation research.

It is hard to generalise when discussing formal proofs, as each proof is different from the others: each problem uses its own vocabulary and notation, each property requires a different proof strategy to be proved and each venue has different requirements on proof structure and the use of natural language. Nevertheless, this module has identified the choices to be made when developing a proof, the set of possible options and the information that should be considered in each decision. Examples of formal proofs, tools providing support for proof writing and Internet resources have also been presented.

A final piece of advice on learning formal proofs is: learn from experience. Writing our own proofs (with the feedback of an expert) and reading proofs critically (understanding and evaluating them, and comparing them to other proofs) is the best way to develop strong skills in formal proof.

Self-assessment exercises

1. Using the inference and equivalence rules of propositional logic, which of the following statements is a contradiction? (*i.e.* it allows the deduction of "false")

a) $P \rightarrow \neg P$
b) $P \wedge (Q \vee (\neg P))$
c) $P \rightarrow$ false
d) $P \leftrightarrow (\neg P)$

2. An integer n is a perfect square if there is an integer k such that $n = k \cdot k = k^2$. Prove that if n is a perfect square, then the remainder of dividing n by 4 is either 0 or 1 ($n \equiv 0 \bmod 4$ or $n \equiv 1 \bmod 4$).

3. Prove that in any group of 8 or more people, at least two were born on the same day of the week.

4. Prove that the square of any integer number is larger or equal than the original ($\forall x \in \mathbb{Z} : x^2 \geq x$). Is there any premise that allows to ensure that it is strictly larger, *i.e.* $x^2 > x$?

Solutions

1.a) Statement (a) is equivalent to $(\neg P)$ and therefore is not a contradiction.

$$
\begin{aligned}
P \rightarrow (\neg P) \equiv & \quad \text{Using the equivalence } (A \rightarrow B) \equiv ((\neg A) \vee B) \\
(\neg P) \vee (\neg P) \equiv & \quad \text{Using the equivalence } (A \vee A) \equiv A \\
(\neg P) &
\end{aligned}
$$

1.b) Statement (b) is equivalent to $(P \wedge Q)$ and therefore is not a contradiction.

$$
\begin{aligned}
P \wedge (Q \vee (\neg P)) \equiv & \quad \text{Using the equivalence } (A \wedge (B \vee C)) \equiv ((A \wedge B) \vee (A \wedge C)) \\
(P \wedge Q) \vee (P \wedge (\neg P)) \equiv & \quad \text{Using the equivalence } (A \wedge (\neg A)) \equiv \text{false} \\
(P \wedge Q) \vee \text{false} \equiv & \quad \text{Using the equivalence } (A \vee \text{false}) \equiv A \\
(P \wedge Q) &
\end{aligned}
$$

1.c) Statement (c) is equivalent to $(\neg P)$ and therefore is not a contradiction.

$$
\begin{aligned}
P \rightarrow \text{false} \equiv & \quad \text{Using the equivalence } (A \rightarrow B) \equiv (\neg A \vee B) \\
(\neg P) \vee \text{false} \equiv & \quad \text{Using the equivalence } (A \vee \text{false}) \equiv A \\
(\neg P) &
\end{aligned}
$$

1.d) Statement (d) is a contradiction.

$$
\begin{aligned}
P \leftrightarrow (\neg P) \equiv & \quad \text{Using the equivalence } (A \leftrightarrow B) \equiv ((A \rightarrow B) \wedge (B \rightarrow A)) \\
(P \rightarrow (\neg P)) \wedge ((\neg P) \rightarrow P) \equiv & \quad \text{Using the equivalence } (A \rightarrow B) \equiv (\neg A \vee B) \\
((\neg P) \vee (\neg P)) \wedge (\neg(\neg P) \vee P) \equiv & \quad \text{Using the equivalence } (\neg(\neg A)) \equiv A \\
((\neg P) \vee (\neg P)) \wedge (P \vee P) \equiv & \quad \text{Using the equivalence } (A \vee A) \equiv A \\
(\neg P) \wedge P \equiv & \quad \text{Using the equivalence } (A \wedge (\neg A)) \equiv \text{false} \\
\text{false} &
\end{aligned}
$$

2. We use proof by cases, considering the potential remainders of the division of k by 4. There are four potential remainders of a division by 4: 0 ($k \equiv 0 \bmod 4$), 1 ($k \equiv 1 \bmod 4$), 2 ($k \equiv 2 \bmod 4$) or 3 ($k \equiv 3 \bmod 4$).

- **Case ($k \equiv 0 \bmod 4$):** k can be rewritten as $4 \cdot q$ for some integer q, the quotient of the division by 4. Then, $n = k^2$ can be expressed as: $n = k^2 = (4 \cdot q)^2 = 4^2 \cdot q^2 = 4 \cdot (4 \cdot q^2)$. Therefore, $n \equiv 0 \bmod 4$ as n is a multiple of 4.

- **Case ($k \equiv 1 \bmod 4$):** k can be rewritten as $4 \cdot q + 1$. Then, $n = k^2$ can be rewritten as: $n = k^2 = (4 \cdot q + 1)^2 = (4q)^2 + 1^2 + 2 \cdot (4q) = 4(4q^2 + 2q) + 1$. Therefore, $n \equiv 1 \bmod 4$ as n is a multiple of 4 plus 1.

- **Case ($k \equiv 2 \bmod 4$):** k can be rewritten as $4 \cdot q + 2$. Then, $n = k^2$ can be rewritten as: $n = k^2 = (4 \cdot q + 2)^2 = (4q)^2 + 2^2 + 2 \cdot 2 \cdot (4q) = 4(4q^2 + 4q + 1)$. Therefore, $n \equiv 0 \bmod 4$ as n is a multiple of 4.

- **Case ($k \equiv 3 \bmod 4$):** k can be rewritten as $4 \cdot q + 3$. Then, $n = k^2$ can be rewritten as: $n = k^2 = (4 \cdot q + 3)^2 = (4q)^2 + 3^2 + 2 \cdot 3 \cdot (4q) = 4^2q^2 + 9 + 6 \cdot 4q = 4^2q^2 + (8+1) + 6 \cdot = 4(4q^2 + 6q + 2) + 1$. Therefore, $n \equiv 1 \bmod 4$ as n is a multiple of 4 plus 1.

We have considered all the potential remainders of k modulo 4, and for each of these cases, we have seen that the remainder of dividing $n = k^2$ by 4 is either 0 or 1. ■

3. We can prove this property using the pigeonhole principle. There are at least 8 people in the group, and there are only 7 seven weekdays (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday). By considering that people are the pigeons of the principle and weekdays are the pigeonholes, there will be at least one weekday having two people assigned. ■

4. We use proof by cases on the sign of the integer x (positive, negative or zero). Each case can be proved directly using direct proof:

- If x is zero, then $x^2 = 0 \cdot 0 = 0$.

- If x is negative, then x^2 is the product of two negative numbers, which will be positive. Any positive number is greater than a negative number.

- If x is positive, then $x^2 = x \cdot x = x \cdot (x - 1 + 1) = x + x \cdot (x - 1)$. We have that $(x - 1)$ is either zero or a positive number, and x is positive by definition of this case. Therefore, their product will either be zero (if $x - 1$ is 0) or a positive integer. This means that $x + x \cdot (x - 1)$ will either be equal to x or strictly larger than x.

Regarding the premises that can ensure that $x^2 > x$, we see that the equality appears when $x = 0$ or $x = 1$ ($x - 1 = 0$ from the last case). Therefore, if $x \neq 0$ and $x \neq 1$, then $x^2 > x$. ■

Glossary

absurd *See* contradiction.

axiom Statement that is accepted as true without need for a proof. Axioms are the accepted foundations of a given logic theory. For example, an axiom of the theory of equality is reflexivity, *i.e.* "any number is equal to itself". The set of axioms should be minimal, that is, axioms should not be provable from other axioms and the chosen logic.

claim *See* conjecture.

conjecture Statement whose validity is supported by partial evidence (available empirical results, extrapolation from similar problems, current accepted theories, ...) but which has not been confirmed through complete experimentation or formal proof.

contradiction Set of statements that cannot be simultaneously satisfied, either because one of them is known to be false (*e.g.* it is the negation of an axiom like $1 \neq 1$ or $0 = 1$) or because they are logically incompatible between them (*e.g.* one statement is the negation of another, $A \wedge \neg A$).

corollary Statement that is a consequence of a previous statement.

counterexample Scenario in which a property is not satisfied.

e.g. Abbreviation of "*exempli gratia*" (latin for "for example"), which is commonly used in mathematical and scientific texts to introduce an example.

fallacy Incorrect application of an inference rule which leads to an invalid conclusion. For example, "if A implies B and B holds, then A holds" is a fallacy: B may be true even if A is not, the implication goes only on one direction.

hypothesis Potential explanation of a phenomenon. In logic, a hypothesis is an assumption introduced into a logical argument in order to be validated or refuted. For example, proof by contradiction considers the negation of our theorem as the hypothesis and then tries to refute this hypothesis. This term can also be used as a synonym of "conjecture", if there is no proof or refutation of the hypothesis.

i.e. Abbreviation of "*id est*" (latin for "that is"), which is commonly used in mathematical and scientific texts to provide a clarification.

inference rule Logic rule that describes how to build an statement from axioms and premises. It establishes the valid conclusions of a set of statements. A classical example of inference rule is *modus ponens*: "if A implies B and A holds, then B also holds" ($A \rightarrow B, A \vdash B$).

invariant Property that is satisfied in all possible configurations of a system. For example, in a computer program, a loop invariant is a property that is satisfied in every iteration of the loop.

lemma Statement that describes an auxiliary property or intermediate result within a proof.

open problem Property that has not been proved or disproved yet, despite repeated attempts. Some open problems have been defined centuries ago and there is still no known way to prove or disprove them.

paradox Logical argument in which an initial hypothesis and a sequence of deduction steps apparently lead to a contradiction. There are several explanations for the paradox: either the initial hypothesis is false, one of the deduction steps is incorrect or the contradiction is only apparent.

premise Statement that will be used to reach the conclusion of the proof, *i.e.* an intermediate statement in our proof.

proposition *See* statement.

q.e.d. Abbreviation of "*quod erat demonstrandum*" (latin for "that which was to be proved"). This abbreviation is located at the end of a proof, indicating that the goal has been proved in the last step. It is equivalent to the mathematical tombstone symbol ■ or □.

statement Assertion that can be either true or false.

theorem Final statement of a proof which is deduced from axioms and intermediate statements by applying a set of inference rules.

Bibliography

Recommended reading

Briony J. Oates (2006) *Researching Information Systems and Computing*. SAGE Publications.

Antonella Cupillari (2005) *The Nuts and Bolts of Proofs*. Elsevier Academic Press, 3rd edition.

Daniel J. Velleman (1994) *How to Prove It: A Structured Approach*. Cambridge University Press, 2nd edition.

Daniel Solow (2004) *How to Read and Do Proofs: An Introduction to Mathematical Thought Processes*. Wiley, 4th edition.

Ted Sundstrom (2006) *Mathematical Reasoning: Writing and Proof*. Prentice Hall, 2nd edition.

Additional reading

Hodges W. (1991) *Logic: an introduction to elementary logic*. Penguin books.

Keith J. Devlin (2002) *The Millenium Problems: The seven greatest unsolved mathematical puzzles of our time*. Basic Books.

William Dunham (1991) *Journey through Genius: The great theorems of Mathematics*. Penguin.