The Use of Logic: Implementing Propositional Reasoning and Proof Construction

Jan van Eijck

June 7, 2012

Abstract

The purpose of this lecture is to demonstrate how propositional reasoning can be implemented, and to point out why quantifier reasoning in general cannot. This should lead to a clearer understanding of the limitations of computers as tools for formal/mathematical reasoning. Also, it should provide motivation for developing our own reasoning skills.

Haskell versions of ‘not’, ‘and’ and ‘or’

We start with implementations of the Boolean connectives.

```
not            :: Bool -> Bool
not True        = False
not False       = True

(&&)   :: Bool -> Bool -> Bool
False && x    = False
True  && x    = x

(||)   :: Bool -> Bool -> Bool
False || x    = x
True  || x    = True
```

Definitions of $\Rightarrow$ and exclusive disjunction

```
infix 6 ==>

(==>) :: Bool -> Bool -> Bool
True  ==> x = x
False ==> x = True

infixr 2 <+>

(<+>) :: Bool -> Bool -> Bool
x <+> y = x /= y
```

Definition of example Formulas

```
form1 p q = p ==> (q ==> p)
form2 p q = (p ==> q) ==> p
```

Definition of example Formulas

```
form1 p q = p ==> (q ==> p)
form2 p q = (p ==> q) ==> p
```

Alternative Definition with Lambda Abstraction

```
form1 = \ p q -> p ==> (q ==> p)
form2 = \ p q -> (p ==> q) ==> p
```

Definition of example Formulas

```
form1 p q = p ==> (q ==> p)
form2 p q = (p ==> q) ==> p
```

Alternative Definition with Lambda Abstraction

```
form1 = \ p q -> p ==> (q ==> p)
form2 = \ p q -> (p ==> q) ==> p
```

Read this as $\lambda p \lambda q.(p \Rightarrow (q \Rightarrow p))$ and $\lambda p \lambda q.((p \Rightarrow q) \Rightarrow p)$.

Logical Validity

Logical Validity

A propositional formula P is logically valid if it is receives the value **true** for all truth values of its proposition letters.

Logical Validity

A propositional formula P is logically valid if it is receives the value **true** for all truth values of its proposition letters.

To implement this, we have to distinguish propositional formulas according to the number of different proposition letters they contain.

Logical Validity

A propositional formula P is logically valid if it is receives the value **true** for all truth values of its proposition letters.

To implement this, we have to distinguish propositional formulas according to the number of different proposition letters they contain.

We *abstract* over the different proposition letters, and the type of the lambda term depends on the number of proposition letters.

Logical Validity

A propositional formula P is logically valid if it is receives the value **true** for all truth values of its proposition letters.

To implement this, we have to distinguish propositional formulas according to the number of different proposition letters they contain.

We *abstract* over the different proposition letters, and the type of the lambda term depends on the number of proposition letters.

A propositional formula with two different proposition letters has type `Bool -> Bool -> Bool`, one with three different proposition letters has type `Bool -> Bool -> Bool -> Bool`, and so on.

```
valid1 :: (Bool -> Bool) -> Bool
valid1 bf =  (bf True) && (bf False)

valid2 :: (Bool -> Bool -> Bool)  -> Bool
valid2 bf = and [bf p q | p <- [True,False],
                          q <- [True,False]]

valid3 :: (Bool -> Bool -> Bool -> Bool) -> Bool
valid3 bf = and [ bf p q r | p <- [True,False],
                             q <- [True,False],
                             r <- [True,False]]
```

Trying it Out

```
form1 p q = p ==> (q ==> p)
form2 p q = (p ==> q) ==> p
```

```
GSWH> :t form1
form1 :: Bool -> Bool -> Bool
GSWH> valid2 form1
True
GSWH> valid2 form2
False
```

Logical Equivalence

Propositional formulas are logically equivalent if they get the same truth values, no matter what the truth values are of the proposition letters.

```
logEquiv1 ::  (Bool -> Bool) ->
                      (Bool -> Bool) -> Bool
logEquiv1 bf1 bf2 =  (bf1 True  == bf2 True)
                  && (bf1 False == bf2 False)

logEquiv2 :: (Bool -> Bool -> Bool) ->
                (Bool -> Bool -> Bool) -> Bool
logEquiv2 bf1 bf2 = and [(bf1 p q) == (bf2 p q)
                           |  p <- [True,False],
                              q <- [True,False]]
```

```
logEquiv3 :: (Bool -> Bool -> Bool -> Bool) ->
               (Bool -> Bool -> Bool -> Bool) -> Bool

logEquiv3 bf1 bf2 =
      and [(bf1 p q r) == (bf2 p q r)
                       |  p <- [True,False],
                          q <- [True,False],
                          r <- [True,False]]
```

Trying it Out

```
formula1 p q = (not p) && (p ==> q)
formula2 p q = not (q && (not p))
formula3 p q = p
formula4 p q = (p <+> q) <+> q
formula5 p q = p <+> (q <+> q)
```

```
Main> valid2 (\ p q -> (formula1 p q == formula2 p q))
False
Main> logEquiv2 formula1 formula2
False
Main> logEquiv2 formula3 formula4
True
```

Useful propositional equivalences

Useful propositional equivalences

1. $P \equiv \neg\neg P$ (law of double negation),

Useful propositional equivalences

1. $P \equiv \neg\neg P$ (law of double negation),
2. $P \wedge P \equiv P$; $P \vee P \equiv P$ (laws of idempotence),

Useful propositional equivalences

1. $P \equiv \neg\neg P$ (law of double negation),
2. $P \wedge P \equiv P$; $P \vee P \equiv P$ (laws of idempotence),
3. $(P \Rightarrow Q) \equiv \neg P \vee Q$;

   $\neg(P \Rightarrow Q) \equiv P \wedge \neg Q$,

Useful propositional equivalences

1. $P \equiv \neg\neg P$ (law of double negation),
2. $P \wedge P \equiv P$; $P \vee P \equiv P$ (laws of idempotence),
3. $(P \Rightarrow Q) \equiv \neg P \vee Q$;

   $\neg(P \Rightarrow Q) \equiv P \wedge \neg Q$,
4. $(\neg P \Rightarrow \neg Q) \equiv (Q \Rightarrow P)$;

   $(P \Rightarrow \neg Q) \equiv (Q \Rightarrow \neg P)$;

   $(\neg P \Rightarrow Q) \equiv (\neg Q \Rightarrow P)$ (laws of contraposition),

Useful propositional equivalences

1. $P \equiv \neg\neg P$ (law of double negation),
2. $P \wedge P \equiv P$; $P \vee P \equiv P$ (laws of idempotence),
3. $(P \Rightarrow Q) \equiv \neg P \vee Q$;

   $\neg(P \Rightarrow Q) \equiv P \wedge \neg Q$,
4. $(\neg P \Rightarrow \neg Q) \equiv (Q \Rightarrow P)$;

   $(P \Rightarrow \neg Q) \equiv (Q \Rightarrow \neg P)$;

   $(\neg P \Rightarrow Q) \equiv (\neg Q \Rightarrow P)$ (laws of contraposition),
5. $$\begin{aligned} (P \Leftrightarrow Q) &\equiv [(P \Rightarrow Q) \wedge (Q \Rightarrow P)] \\ &\equiv [(P \wedge Q) \vee (\neg P \wedge \neg Q)], \end{aligned}$$

Useful propositional equivalences

1. $P \equiv \neg\neg P$ (law of double negation),
2. $P \wedge P \equiv P$; $P \vee P \equiv P$ (laws of idempotence),
3. $(P \Rightarrow Q) \equiv \neg P \vee Q$;
   $\neg(P \Rightarrow Q) \equiv P \wedge \neg Q$,
4. $(\neg P \Rightarrow \neg Q) \equiv (Q \Rightarrow P)$;
   $(P \Rightarrow \neg Q) \equiv (Q \Rightarrow \neg P)$;
   $(\neg P \Rightarrow Q) \equiv (\neg Q \Rightarrow P)$ (laws of contraposition),
5. $$\begin{aligned}(P \Leftrightarrow Q) &\equiv [(P \Rightarrow Q) \wedge (Q \Rightarrow P)] \\ &\equiv [(P \wedge Q) \vee (\neg P \wedge \neg Q)],\end{aligned}$$
6. $P \wedge Q \equiv Q \wedge P$; $P \vee Q \equiv Q \vee P$ (laws of commutativity),

7. $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$;

$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$ (DeMorgan laws).

7. $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$;

   $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$ (DeMorgan laws).

8. $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$;

   $P \vee (Q \vee R) \equiv (P \vee Q) \vee R$ (laws of associativity),

7. $\neg(P \land Q) \equiv \neg P \lor \neg Q$;

   $\neg(P \lor Q) \equiv \neg P \land \neg Q$ (DeMorgan laws).

8. $P \land (Q \land R) \equiv (P \land Q) \land R$;

   $P \lor (Q \lor R) \equiv (P \lor Q) \lor R$ (laws of associativity),

9. $P \land (Q \lor R) \equiv (P \land Q) \lor (P \land R)$;

   $P \lor (Q \land R) \equiv (P \lor Q) \land (P \lor R)$ (distribution laws),

7. $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$;
   $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$ (DeMorgan laws).

8. $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$;
   $P \vee (Q \vee R) \equiv (P \vee Q) \vee R$ (laws of associativity),

9. $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$;
   $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$ (distribution laws),

You can check all of these by computer, using the implementations of `logEquiv1`, `logEquiv2`, and `logEquiv3`.

Useful Quantifier Equivalences

Useful Quantifier Equivalences

1. $\forall x \forall y \Phi(x,y) \equiv \forall y \forall x \Phi(x,y)$;

   $\exists x \exists y \Phi(x,y) \equiv \exists y \exists x \Phi(x,y)$,

Useful Quantifier Equivalences

1. $\forall x \forall y \Phi(x,y) \;\equiv\; \forall y \forall x \Phi(x,y);$

   $\exists x \exists y \Phi(x,y) \;\equiv\; \exists y \exists x \Phi(x,y),$

2. $\neg \forall x \Phi(x) \;\equiv\; \exists x \neg \Phi(x);$

   $\neg \exists x \Phi(x) \;\equiv\; \forall x \neg \Phi(x);$

   $\neg \forall x \neg \Phi(x) \;\equiv\; \exists x \Phi(x);$

   $\neg \exists x \neg \Phi(x) \;\equiv\; \forall x \Phi(x),$

Useful Quantifier Equivalences

1. $\forall x \forall y \Phi(x,y) \;\equiv\; \forall y \forall x \Phi(x,y)$;

   $\exists x \exists y \Phi(x,y) \;\equiv\; \exists y \exists x \Phi(x,y)$,

2. $\neg \forall x \Phi(x) \;\equiv\; \exists x \neg \Phi(x)$;

   $\neg \exists x \Phi(x) \;\equiv\; \forall x \neg \Phi(x)$;

   $\neg \forall x \neg \Phi(x) \;\equiv\; \exists x \Phi(x)$;

   $\neg \exists x \neg \Phi(x) \;\equiv\; \forall x \Phi(x)$,

3. $\forall x (\Phi(x) \wedge \Psi(x)) \;\equiv\; (\forall x \Phi(x) \wedge \forall x \Psi(x))$;

   $\exists x (\Phi(x) \vee \Psi(x)) \;\equiv\; (\exists x \Phi(x) \vee \exists x \Psi(x))$.

Useful Quantifier Equivalences

1. $\forall x \forall y \Phi(x,y) \ \equiv\ \forall y \forall x \Phi(x,y);$

   $\exists x \exists y \Phi(x,y) \ \equiv\ \exists y \exists x \Phi(x,y),$

2. $\neg \forall x \Phi(x) \ \equiv\ \exists x \neg \Phi(x);$

   $\neg \exists x \Phi(x) \ \equiv\ \forall x \neg \Phi(x);$

   $\neg \forall x \neg \Phi(x) \ \equiv\ \exists x \Phi(x);$

   $\neg \exists x \neg \Phi(x) \ \equiv\ \forall x \Phi(x),$

3. $\forall x (\Phi(x) \wedge \Psi(x)) \ \equiv\ (\forall x \Phi(x) \wedge \forall x \Psi(x));$

   $\exists x (\Phi(x) \vee \Psi(x)) \ \equiv\ (\exists x \Phi(x) \vee \exists x \Psi(x)).$

There is no mechanical method (like the truth table method) for checking logical equivalence of quantified formulas. That's why we need to develop skills in proving mathematical statements . . .

Quantifiers as Procedures

```
any, all        :: (a -> Bool) -> [a] -> Bool
any p            = or  . map p
all p            = and . map p
```

To understand the implementations of `all` and `any`, one has to know that `or` and `and` are the generalizations of (inclusive) disjunction and conjunction to lists. They have type `[Bool] -> Bool`.

Saying that all elements of a list `xs` satisfy a property `p` boils down to: the list `map p xs` contains only `True`.

Similarly, saying that some element of a list `xs` satisfies a property `p` boils down to: the list `map p xs` contains at least one `True`.

This explains the implementation of `all`: first apply `map p`, next apply `and`.

In the case of `any`: first apply `map p`, next apply `or`.

This explains the implementation of `all`: first apply `map p`, next apply `and`.

In the case of `any`: first apply `map p`, next apply `or`.

The action of applying a function `g :: b -> c` **after** a function `f :: a -> b` is performed by the function `g . f :: a -> c` , the composition of `f` and `g`.

The definitions of `all` and `any` are used as follows:

```
Prelude> any (<3) [0..]
True
Prelude> all (<3) [0..]
False
Prelude>
```

The functions `forall` and `exists` get us even closer to standard logical notation. These functions are like `all` and `any`, but they first take the restriction argument, next the body:

```
forall, exists :: [a] -> (a -> Bool) -> Bool
forall xs p = all p xs
exists xs p = any p xs
```

Now, e.g., the formula $\forall x \in \{1,4,9\} \exists y \in \{1,2,3\}\ x = y^2$ can be implemented as a test, as follows:

```
qform = forall
           [1,4,9]
             (\ x ->
                exists [1,2,3] (\ y -> x == y^2))
```

Now, e.g., the formula $\forall x \in \{1,4,9\} \exists y \in \{1,2,3\}\ x = y^2$ can be implemented as a test, as follows:

```
qform = forall
           [1,4,9]
             (\ x ->
                exists [1,2,3] (\ y -> x == y^2))
```

```
TAMO> qform
True
```

But caution: the implementations of the quantifiers are procedures, not algorithms.

A call to `all` or `any` (or `forall` or `exists`) need not terminate.

The call `forall [0..] (>=0)` will run forever. This illustrates once more that the quantifiers are in essence more complex than the propositional connectives . . .

How to present a proof: general

How to present a proof: general

1. Write correct English, try to express yourself clearly.

How to present a proof: general

1. Write correct English, try to express yourself clearly.
2. Make sure the reader knows exactly what you are up to.

How to present a proof: general

1. Write correct English, try to express yourself clearly.
2. Make sure the reader knows exactly what you are up to.
3. Say what you mean when introducing a variable.

How to present a proof: general

1. Write correct English, try to express yourself clearly.
2. Make sure the reader knows exactly what you are up to.
3. Say what you mean when introducing a variable.
4. Don't start a sentence with symbols, don't write formulas only.

How to present a proof: general

1. Write correct English, try to express yourself clearly.
2. Make sure the reader knows exactly what you are up to.
3. Say what you mean when introducing a variable.
4. Don't start a sentence with symbols, don't write formulas only.
5. Use words like *'thus'*, *'therefore'*, *'hence'*, etc. to link up your formulas.

How to present a proof: general

1. Write correct English, try to express yourself clearly.
2. Make sure the reader knows exactly what you are up to.
3. Say what you mean when introducing a variable.
4. Don't start a sentence with symbols, don't write formulas only.
5. Use words like *'thus'*, *'therefore'*, *'hence'*, etc. to link up your formulas.
6. Be relevant and succinct.

How to present a proof: specific

How to present a proof: specific

- When constructing proofs, use the following schema:

  Given: ...
  To be proved: ...
  Proof: ...

How to present a proof: specific

7 When constructing proofs, use the following schema:

> **Given:** . . .
> **To be proved:** . . .
> **Proof:** . . .

8 Look up definitions of defined notions, and use these definitions to re-write both *Given* and *To be proved*.

How to present a proof: specific

7 When constructing proofs, use the following schema:

> **Given:** . . .
> **To be proved:** . . .
> **Proof:** . . .

8 Look up definitions of defined notions, and use these definitions to re-write both *Given* and *To be proved*.

9 Make sure you have a sufficient supply of scratch paper, make a fair copy of the end-product —whether you think it to be faultless or not.

How to present a proof: specific

7. When constructing proofs, use the following schema:

    Given: . . .
    To be proved: . . .
    Proof: . . .

8. Look up definitions of defined notions, and use these definitions to re-write both *Given* and *To be proved*.

9. Make sure you have a sufficient supply of scratch paper, make a fair copy of the end-product —whether you think it to be faultless or not.

10. Ask yourself two things: Is this correct? Can others read it?

Proof Recipes: Subproofs

Given: A, B, $\ldots$

To be proved: P

Proof:

$\ldots$

> *Suppose* C $\ldots$
>
> *To be proved:* Q
>
> *Proof:* $\ldots$
>
> $\ldots$
>
> Thus Q

$\ldots$

Thus P

Scope of Assumptions

The purpose of ‘Suppose’ is to add a new given to the list of assumptions that may be used, but only for the duration of the subproof of which ‘Suppose’ is the head.

Scope of Assumptions

The purpose of ‘Suppose’ is to add a new given to the list of assumptions that may be used, but only for the duration of the subproof of which ‘Suppose’ is the head.

If the current list of givens is $P_1, \ldots, P_n$ then ‘Suppose Q’ extends this list to $P_1, \ldots, P_n, Q$.

Scope of Assumptions

The purpose of ‘Suppose’ is to add a new given to the list of assumptions that may be used, but only for the duration of the subproof of which ‘Suppose’ is the head.

If the current list of givens is $P_1, \ldots, P_n$ then ‘Suppose Q’ extends this list to $P_1, \ldots, P_n, Q$.

In general, inside a box, you can use all the givens and assumptions of all the including boxes. Thus, in the innermost box of the example, the givens are A, B, C. This illustrates the importance of indentation for keeping track of the ‘current box’.

Scope of Assumptions

The purpose of ‘Suppose’ is to add a new given to the list of assumptions that may be used, but only for the duration of the subproof of which ‘Suppose’ is the head.

If the current list of givens is $P_1, \ldots, P_n$ then ‘Suppose Q’ extends this list to $P_1, \ldots, P_n, Q$.

In general, inside a box, you can use all the givens and assumptions of all the including boxes. Thus, in the innermost box of the example, the givens are A, B, C. This illustrates the importance of indentation for keeping track of the ‘current box’.

Attitude Grasp the importance of proper formatting. Use indentation to clarify the structure of your proofs. By getting it on the paper in a structured way, you will clear up confusion in your mind.

The two ways of encountering a logical symbol

1. The symbol can appear in the given, or in an assumption.
2. The symbol can appear in the statement that is to be proved.

The two ways of encountering a logical symbol

1. The symbol can appear in the given, or in an assumption.
2. The symbol can appear in the statement that is to be proved.

In the first case the rule to use is an *elimination* rule, in the second case an *introduction* rule.

The two ways of encountering a logical symbol

1. The symbol can appear in the given, or in an assumption.
2. The symbol can appear in the statement that is to be proved.

In the first case the rule to use is an *elimination* rule, in the second case an *introduction* rule.

Elimination rules enable you to reduce a proof problem to a new, hopefully simpler, one.

The two ways of encountering a logical symbol

1. The symbol can appear in the given, or in an assumption.
2. The symbol can appear in the statement that is to be proved.

In the first case the rule to use is an *elimination* rule, in the second case an *introduction* rule.

Elimination rules enable you to reduce a proof problem to a new, hopefully simpler, one.

Introduction rules make clear how to prove a goal of a certain given shape.

Introduction of an Implication

Given: $\ldots$
To be proved: $\Phi \Rightarrow \Psi$
Proof:

> *Suppose* Φ
> *To be proved:* Ψ
> *Proof:* $\ldots$

Thus $\Phi \Rightarrow \Psi$.

Use (Elimination) of an implication

This rule is also called *Modus Ponens*.

Given: $\Phi \Rightarrow \Psi$, Φ
Thus Ψ.

Example of Reasoning with Implication

Given: $P \Rightarrow Q$, $Q \Rightarrow R$

To be proved: $P \Rightarrow R$

Proof:

> *Suppose* P
>
> *To be proved:* R
>
> *Proof:* From $P \Rightarrow Q$ and P, conclude Q.
>
> Next, from $Q \Rightarrow R$ and Q, conclude R.

Thus $P \Rightarrow R$

How to prove an implication

If the ‘to be proved’ is an implication $\Phi \Rightarrow \Psi$, then your proof should start with the following Obligatory Sentence:

Suppose that Φ holds.

The obligatory first sentence accomplishes the following things (cf. the 2nd commandment above).

- It informs the reader that you are going to apply the Deduction Rule in order to establish that $\Phi \Rightarrow \Psi$.
- The reader also understands that it is now Ψ that you are going to derive (instead of $\Phi \Rightarrow \Psi$).
- Thus, starting with the obligatory sentence informs the reader in an efficient way about your plans.

Example

Assume that $n, m \in \mathbb{N}$.

To show: $(m$ is even $\wedge$ n is even$) \Rightarrow m + n$ is even.

Detailed proof:

> Assume that $(m$ even $\wedge$ n even$)$.
> Then ($\wedge$-elimination) m and n are both even.
> For instance, $p, q \in \mathbb{N}$ exist such that $m = 2p$, $n = 2q$.
> Then $m + n = 2p + 2q = 2(p + q)$ is even.

Thus $(m$ is even $\wedge$ n is even$) \Rightarrow m + n$ is even.

Example

Assume that $n, m \in \mathbb{N}$.

To show: $(m \text{ is even} \wedge n \text{ is even}) \Rightarrow m + n \text{ is even}$.

Detailed proof:

> Assume that $(m \text{ even} \wedge n \text{ even})$.
> Then ($\wedge$-elimination) m and n are both even.
> For instance, $p, q \in \mathbb{N}$ exist such that $m = 2p$, $n = 2q$.
> Then $m + n = 2p + 2q = 2(p + q)$ is even.

Thus $(m \text{ is even} \wedge n \text{ is even}) \Rightarrow m + n \text{ is even}$.

Concise version:

Assume that m and n are even.
For instance, $m = 2p$, $n = 2q$, $p, q \in \mathbb{N}$.
Then $m + n = 2p + 2q = 2(p + q)$ is even.

Using and Proving Conjunctions

- Elimination of $\wedge$: Use the given $P \wedge Q$ by using both P and Q.

Using and Proving Conjunctions

- Elimination of $\wedge$: Use the given $P \wedge Q$ by using both P and Q.
- Introduction of $\wedge$: Show a result of the form $P \wedge Q$ by first proving P, next proving Q.

Using and Proving Negations

Introduction of $\neg$:

Given: . . .
To be proved: $\neg\Phi$
Proof:

    Suppose Φ
    To be proved: $\perp$
    Proof: . . .

Thus $\neg\Phi$.

Using and Proving Negations

Introduction of $\neg$:

Given: $\ldots$
To be proved: $\neg\Phi$
Proof:

> *Suppose* Φ
> *To be proved:* $\perp$
> *Proof:* $\ldots$

Thus $\neg\Phi$.

Elimination of $\neg$:

Given: Φ, $\neg\Phi$
Thus Ψ.

General advice: try to move negation symbols inward as much as possible before treating them.

Example: there are infinitely many prime numbers

Example: there are infinitely many prime numbers

Suppose there are only finitely many prime numbers, and $p_1, \ldots, p_n$ is a list of all primes. Consider the number $m = (p_1 p_2 \cdots p_n) + 1$. Note that m is not divisible by p_1, for dividing m by p_1 gives quotient $p_2 \cdots p_n$ and remainder 1. Similarly, division by $p_2, p_3, \ldots$ always gives a remainder 1.

Example: there are infinitely many prime numbers

Suppose there are only finitely many prime numbers, and $p_1, \ldots, p_n$ is a list of all primes. Consider the number $m = (p_1 p_2 \cdots p_n) + 1$. Note that m is not divisible by p_1, for dividing m by p_1 gives quotient $p_2 \cdots p_n$ and remainder 1. Similarly, division by $p_2, p_3, \ldots$ always gives a remainder 1.

- $LD(m)$ is prime,
- For all $i \in \{1, \ldots n\}$, $LD(m) \neq p_i$.

Thus, we have found a prime number $LD(m)$ different from all the prime numbers in our list $p_1, \ldots, p_n$, contradicting the assumption that $p_1, \ldots, p_n$ was the full list of prime numbers.

The assumption that there are only a finite number of primes leads to a contradiction. Thus, there are infinitely many prime numbers.

Example: There is no rational number x with $x^2 = 2$.

Example: There is no rational number x with $x^2 = 2$.

Assume there is a number $x \in \mathbb{Q}$ with $x^2 = 2$. Then there are $m, n \in \mathbb{N}$, $n \neq 0$ with $(m/n)^2 = 2$. We can further assume that m/n is cancelled down to its lowest form, i.e., there are no $k, p, q \in \mathbb{Z}$ with $k \neq 1$, $m = kp$ and $n = kq$.

Example: There is no rational number x with $x^2 = 2$.

Assume there is a number $x \in \mathbb{Q}$ with $x^2 = 2$. Then there are $m, n \in \mathbb{N}$, $n \neq 0$ with $(m/n)^2 = 2$. We can further assume that m/n is cancelled down to its lowest form, i.e., there are no $k, p, q \in \mathbb{Z}$ with $k \neq 1$, $m = kp$ and $n = kq$.

We have: $2 = (m/n)^2 = m^2/n^2$, and multiplying both sides by n^2 we find $2n^2 = m^2$. In other words, m^2 is even, and since squares of odd numbers are always odd, m must be even, i.e., there is a p with $m = 2p$. Substitution in $2n^2 = m^2$ gives $2n^2 = (2p)^2 = 4p^2$, and we find that $n^2 = 2p^2$, which leads to the conclusion that n is also even. But this means that there is a q with $n = 2q$, and we have a contradiction with the assumption that m/n was in lowest form. It follows that there is no number $x \in \mathbb{Q}$ with $x^2 = 2$.

Example: There is no rational number x with $x^2 = 2$.

Assume there is a number $x \in \mathbb{Q}$ with $x^2 = 2$. Then there are $m, n \in \mathbb{N}$, $n \neq 0$ with $(m/n)^2 = 2$. We can further assume that m/n is cancelled down to its lowest form, i.e., there are no $k, p, q \in \mathbb{Z}$ with $k \neq 1$, $m = kp$ and $n = kq$.

We have: $2 = (m/n)^2 = m^2/n^2$, and multiplying both sides by n^2 we find $2n^2 = m^2$. In other words, m^2 is even, and since squares of odd numbers are always odd, m must be even, i.e., there is a p with $m = 2p$. Substitution in $2n^2 = m^2$ gives $2n^2 = (2p)^2 = 4p^2$, and we find that $n^2 = 2p^2$, which leads to the conclusion that n is also even. But this means that there is a q with $n = 2q$, and we have a contradiction with the assumption that m/n was in lowest form. It follows that there is no number $x \in \mathbb{Q}$ with $x^2 = 2$.

Therefore, the square root of 2 is not rational.

Proof by Contradiction ('uit het ongerijmde')

In order to prove Φ, add $\neg\Phi$ as a new given, and attempt to deduce an evidently false statement.

In a schema:

Given: ...
To be proved: Φ
Proof:

> *Suppose* $\neg\Phi$
> *To be proved:* $\perp$
> *Proof:* ...

Thus Φ.

Example of a proof by contradiction

Derive from $\neg Q \Rightarrow \neg P$ that $P \Rightarrow Q$.

Given: $\neg Q \Rightarrow \neg P$
To be proved: $P \Rightarrow Q$
Proof:

> *Suppose* P
> *To be proved:* Q
> *Proof:*
>
> > *Suppose* $\neg Q$.
> > Then from $\neg Q \Rightarrow \neg P$ and $\neg Q$: $\neg P$,
> > and contradiction with P.
>
> Thus Q.

Thus $P \Rightarrow Q$.

Proving and Using Disjunctions

Introduction of $\vee$:

Given: Φ. Thus $\Phi \vee \Psi$.

Given: Ψ. Thus $\Phi \vee \Psi$.

Elimination of $\vee$:

Given: $\Phi \vee \Psi, \ldots$
To be proved: Λ
Proof:

> *Suppose* Φ
> *To be proved:* Λ
> *Proof:* $\ldots$
>
> *Suppose* Ψ
> *To be proved:* Λ
> *Proof:* $\ldots$

Thus Λ.

Example of a proof by case distinction

Given: $x \in (A - C)$.
To be proved: $x \in (A - B) \vee x \in (B - C)$.
Proof:

> Suppose $x \in B$.
> From $x \in (A - C)$ we get $x \notin C$, and therefore $x \in (B - C)$.
> Thus $x \in (A - B) \vee x \in (B - C)$.
>
> Suppose $x \notin B$.
> From $x \in (A - C)$ we get $x \in A$, and so $x \in (A - B)$.
> Thus $x \in (A - B) \vee x \in (B - C)$.

It follows that $x \in (A - B) \vee x \in (B - C)$.

Quantifier Reasoning: Universal Quantification

Introduction of $\forall x$:

Given: …
To be proved: $\forall x E(x)$
Proof:

> *Suppose* c is an arbitrary object.
> *To be proved:* $E(c)$
> *Proof:* …

Thus $\forall x E(x)$

Key: ‘let c be an arbitrary object.’ (= “een willekeurig ding”).

Quantifier Reasoning: Universal Quantification

Introduction of $\forall x$:

Given: …
To be proved: $\forall x E(x)$
Proof:

> *Suppose* c is an arbitrary object.
> *To be proved:* $E(c)$
> *Proof:* …

Thus $\forall x E(x)$

Key: ‘let c be an arbitrary object.’ (= “een willekeurig ding”).

Elimination of $\forall x$: conclude from $\forall x E(x)$ that $E(t)$.

Introduction of restricted universal quantification $\forall x \in A$:

Given: ...
To be proved: $\forall x \in A : \; E(x)$
Proof:

> *Suppose* c is an arbitrary object in A.
> *To be proved:* $E(c)$
> *Proof:* ...

Thus $\forall x \in A : \; E(x)$

Key: 'let c be an arbitrary object in A'.

Introduction of a universal quantifier with an implication:

Given: $\ldots$
To be proved: $\forall x(P(x) \Rightarrow Q(x))$
Proof:

> *Suppose* c is an arbitary object satisfying $P(c)$.
> *To be proved:* $Q(c)$
> *Proof:* $\ldots$

Thus $\forall x(P(x) \Rightarrow Q(x))$

Key: ‘let c be an arbitary object satisfying $P(c)$.’

Quantifier Reasoning: Existential Quantification

Introduction of $\exists x : E(x)$:

Conclude from $E(t)$ that $\exists x : E(x)$.

Quantifier Reasoning: Existential Quantification

Introduction of $\exists x : E(x)$:

Conclude from $E(t)$ that $\exists x : E(x)$.

Elimination of $\exists x : E(x)$:

Given: $\exists x E(x), \ldots$
To be proved: Λ
Proof:

> *Suppose* c is an object satisfying $E(c)$.
> *To be proved:* Λ
> *Proof:* $\ldots$

Thus Λ

For restricted existential quantification, just modify the key to: “suppose c is an object in A that satisfies $E(c)$.”

Refuting conjectures

There are only two kinds of mathematical statements: true statements and false ones. All proof attempts of false statements are bound to fail, of course!

Refuting conjectures

There are only two kinds of mathematical statements: true statements and false ones. All proof attempts of false statements are bound to fail, of course!

So don't try to prove them. Instead, try to refute them.

Refuting conjectures

There are only two kinds of mathematical statements: true statements and false ones. All proof attempts of false statements are bound to fail, of course!

So don't try to prove them. Instead, try to refute them.

A famous conjecture made in 1640 by Pierre de Fermat (1601–1665) is that all numbers of the form

$$2^{2^n} + 1$$

are prime. This holds for $n = 0, 1, 2, 3, 4$, for we have: $2^{2^0} + 1 = 2^1 + 1 = 3$, $2^{2^1} + 1 = 2^2 + 1 = 5$, $2^{2^2} + 1 = 2^4 + 1 = 17$, $2^{2^3} + 1 = 2^8 + 1 = 257$, which is prime, and $2^{2^4} + 1 = 2^{16} + 1 = 65537$, which is prime. Apparently, this is as far as Fermat got.

Here is a Haskell refutation of Fermat's conjecture:

```
GSWH> prime (2^2^5 + 1)
False
```

Primes Again

```
prime :: Integer -> Bool
prime n | n < 1     = error "not a positive integer"
        | n == 1    = False
        | otherwise = ld n == n
  where
   ld n = ldf 2 n
   ldf k n | divides k n = k
           | k^2 > n     = n
           | otherwise   = ldf (k+1) n
   divides d n = rem n d == 0
```

Mersenne Primes

$M_n = 2^n - 1$ sometimes is prime when n is prime. Such primes are called Mersenne primes.

Mersenne Primes

$M_n = 2^n - 1$ sometimes is prime when n is prime. Such primes are called Mersenne primes.

```
mersenne :: [(Integer,Integer)]
mersenne =
   [ (n,2^n-1) | n <- [2..], prime (2^n - 1) ]
```

```
notmersenne :: [(Integer,Integer)]
notmersenne =
   [ (n,2^n-1) | n <- [2..],
                 prime n,
                 not (prime (2^n - 1)) ]
```