

Four Howtos

Jan van Eijck

jve@cwi.nl

January 11, 2006

Abstract

How to Set Up a Proof?

How to Read and Analyse a Proof?

How to Transform a Definition into a Haskell Implementation?

How to Read and Understand a Haskell Function?

How to Set Up a Proof

Advice from [1]:

- Proofs are **highly structured** texts. Be aware of their structure.
- When constructing proofs, use the following schema:

Given: ...

To be proved: ...

Proof: ...

- Look up definitions of defined notions, and use these definitions to re-write both **Given** and **To be proved**.
- Make sure you have a sufficient supply of scrap paper, and make a fair copy of the end-product.
- Ask yourself two things: Is this correct? Can others read it?

Proof Recipes: Subproofs

Given: A, B, \dots

To be proved: P

Proof:

...

Suppose $C \dots$

To be proved: Q

Proof: ...

...

Thus Q

...

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.

If the current list of givens is P_1, \dots, P_n then 'Suppose Q ' extends this list to P_1, \dots, 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 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**.

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: ...

To be proved: $\Phi \Rightarrow \Psi$

Proof:

 Suppose Φ

 To be proved: Ψ

 Proof: ...

Thus $\Phi \Rightarrow \Psi$.

This rule is called the **Deduction Rule**.

Example

Given: R is a transitive relation

To be proved: $(x, y) \in R \circ R \Rightarrow (x, y) \in R$

Proof:

Suppose $(x, y) \in R \circ R$

To be proved: $(x, y) \in R$

Proof: From $(x, y) \in R \circ R$:

there is a z with $(x, z) \in R$, $(z, y) \in R$.

From this, by transitivity of R , $(x, y) \in R$.

Thus $(x, y) \in R \circ R \Rightarrow (x, y) \in R$

Use (Elimination) of an implication

This rule is also called **Modus Ponens**.

Given: $\Phi \Rightarrow \Psi, \Phi$

Thus Ψ .

Examples

Given: $x \in A \Rightarrow x \in B, x \in A.$

Thus, $x \in B.$

Remembering the definition of $A \subseteq B:$

Given: $A \subseteq B, x \in A.$

Thus, $x \in B.$

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:

- 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 and n are both even $\Rightarrow m + n$ is even.

Detailed proof:

Assume 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 and n are both even $\Rightarrow m + n$ 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.

Proving and Using Negations

Introduction of \neg :

Given: ...

To be proved: $\neg\Phi$

Proof:

 Suppose Φ

 To be proved: \perp

 Proof: ...

Thus $\neg\Phi$.

Elimination of \neg :

Given: $\Phi, \neg\Phi$

Thus Ψ .

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

Example: there are infinitely many prime numbers

Suppose there are only finitely many prime numbers, and p_1, \dots, 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, \dots always gives a remainder 1.

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

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

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

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 $P \Rightarrow Q$.

Another example of proof by contradiction

From the proof of the Hennessy-Milner theorem:

Assume $s \rightsquigarrow t$, and let $s \xrightarrow{a} s'$. We have to show that there is a t' with $t \xrightarrow{a} t'$ and $s' \rightsquigarrow t'$.

For a contradiction, assume that there is no such t' .

By the fact that $s \xrightarrow{a} s'$ the formula $\langle a \rangle \top$ holds at s . By $s \rightsquigarrow t$, formula $\langle a \rangle \top$ also holds at t .

Therefore the set $U = \{u \mid t \xrightarrow{a} u\}$ is non-empty.

Since \mathbf{N} is image-finite, U must be finite. Let us say

$$U = \{u_1, \dots, u_n\}.$$

Now for every $u_i \in U$ there must be a ψ_i with $\mathbf{M}, s' \models \psi_i$ and $\mathbf{N}, u_i \not\models \psi_i$.

But from this it follows that:

$$\mathbf{M}, s \models \langle a \rangle (\psi_1 \wedge \cdots \wedge \psi_n) \text{ and } \mathbf{N}, t \not\models \langle a \rangle (\psi_1 \wedge \cdots \wedge \psi_n).$$

This contradicts the given that $s \rightsquigarrow t$.

End of subproof looking for a contradiction.

Thus there is a t' with $t \xrightarrow{a} t'$ and $s' \rightsquigarrow t'$.

Proving and Using Disjunctions

Introduction of \vee :

Given: Φ . Thus $\Phi \vee \Psi$.

Given: Ψ . Thus $\Phi \vee \Psi$.

Elimination of \vee :

Given: $\Phi \vee \Psi, \dots$

To be proved: Λ

Proof:

Suppose Φ

To be proved: Λ

Proof: \dots

Suppose Ψ

To be proved: Λ

Proof: \dots

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").

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: ...

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

Proof:

Suppose c is an arbitrary object satisfying $P(c)$.

To be proved: $Q(c)$

Proof: ...

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

Key: 'let c be an arbitrary object satisfying $P(c)$.'

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: $\exists x E(x), \dots$

To be proved: Λ

Proof:

Suppose c is an object satisfying $E(c)$.

To be proved: Λ

Proof: \dots

Thus Λ

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

Strategic Hints

- Concentrate on **what is to be proved**, not on the **given**.
- Keep good track of the list of assumptions that is available in the **current subproof**.
- Always attempt to transform proof problems into **simpler proof problems** that can be proved using additional assumptions. This can be done with \Rightarrow introduction or \forall introduction.
- Move negations inward, if possible.
- If you can avoid proof by contradiction, avoid it.

Example of 'Concentrate on What is to be Proved'

To be proved: The set of R -classes

$$\{|a|_R \mid a \in A\}$$

of an equivalence relation R on A forms a partition of A .

What are the three properties of partitions?

1. every member of P is non-empty.
2. every element of A belongs to some member of P .
3. different members of P are disjoint.

Check these properties one by one, using the information that we can extract from the fact that R is an equivalence relation, and we have a proof.

Patterns: Equality of Two Sets

Given: ...

To be proved: $A = B$

Proof:

\Rightarrow : Let $x \in A$.

To be proved: $x \in B$

Proof: ...

\Leftarrow : Let $x \in B$.

To be proved: $x \in A$

Proof: ...

Thus $A = B$

Examples

Is it true, for all sets A and B , that $\mathcal{P}(A \cap B) = \mathcal{P}A \cap \mathcal{P}B$?

Give a proof or a counterexample.

Is it true, for all sets A and B , that $\mathcal{P}(A \cup B) = \mathcal{P}A \cup \mathcal{P}B$?

Give a proof or a counterexample.

Patterns: proving iff statements

Prove that a binary relation R is transitive iff $R \circ R \subseteq R$.

\Rightarrow : Assume R is a transitive binary relation on A .

To be proved: $R \circ R \subseteq R$.

Proof. Let $(x, y) \in R \circ R$. We must show that $(x, y) \in R$.

From the definition of \circ :

$\exists z \in A : (x, z) \in R, (z, y) \in R$.

It follows from this and transitivity of R that $(x, y) \in R$.

\Leftarrow : Assume $R \circ R \subseteq R$.

To be proved: R is transitive.

Proof. Let $(x, y) \in R, (y, z) \in R$. Then $(x, z) \in R \circ R$.

Since $R \circ R \subseteq R$, it follows from this that $(x, z) \in R$.

Thus, R is transitive.

Patterns: Proof that A is the smallest set satisfying P

To be proved: A is the smallest set satisfying P

Proof:

To be proved: (i): A satisfies P

Proof: ...

To be proved: (ii): A is smallest among the sets satisfying P

Proof: Let B be a set satisfying P .

To be proved: $A \subseteq B$.

Proof: ...

Thus A is the smallest set satisfying P .

Example: Proof that a Set is a Transitive Closure

Prove that $\mathbf{lfp} (\lambda S. S \cup (S \circ S)) R$ is the transitive closure of R .

$\mathbf{lfp} f c$ is the least fixpoint of the operation f , starting from c .

The transitive closure of a set R is the smallest transitive set that includes R .

Let $f = \lambda S. S \cup (S \circ S)$, and let $T = \mathbf{lfp} f R$.

We must show:

1. $R \subseteq T$,
2. T is transitive,
3. If $R \subseteq S$ and S is transitive, then $T \subseteq S$.

1. f is monotone, i.e., for all arguments X , $X \subseteq f(X)$. In particular, $R \subseteq f(R) \subseteq \dots \subseteq T$. This shows that $R \subseteq T$.
2. We show that T is transitive. Since T is a fixpoint, $T = f(T)$, i.e., $T = T \cup (T \circ T)$. In other words, $T \circ T \subseteq T$. It follows from this that T is transitive by an earlier result.
3. Finally, we show that T is the smallest transitive relation that includes R . Let S be an arbitrary transitive relation with $R \subseteq S$. From transitivity of S it follows that $S \circ S \subseteq S$. Therefore, $S = f(S)$, i.e., S is a fixpoint. Since T is the least fixpoint it follows that $T \subseteq S$.

Example (ctd): From Proof to Implementation

We know from the proof that the following implementation is correct:

```
tc :: Ord a => Rel a -> Rel a
tc = lfp (\ s -> (sort.nub) (s ++ (s @@ s)))
```

Proof by Induction

General structure:

Basis

Induction Hypothesis

Induction Step

Example of Proof by Induction

Prove by induction that if R is an euclidean relation, then

$$R^\sim \circ (R \cup R^\sim)^* \circ R \subseteq R.$$

Basis: $R^\sim \circ R \subseteq R$. This follows from the euclideanness of R .

Induction step. The **induction hypothesis** is that

$$R^\sim \circ (R \cup R^\sim)^n \circ R \subseteq R.$$

We must prove that $R^\sim \circ (R \cup R^\sim)^{n+1} \circ R \subseteq R$.

We are done if we can prove the following two facts:

- (i) $R^\sim \circ R \circ (R \cup R^\sim)^n \circ R \subseteq R$,
- (ii) $R^\sim \circ R^\sim \circ (R \cup R^\sim)^n \circ R \subseteq R$.

(i) Assume

$$(x, y) \in R^\sim \circ R \circ (R \cup R^\sim)^n \circ R.$$

Then there is a z with $(x, z) \in R^\sim \circ R$ and $(z, y) \in (R \cup R^\sim)^n \circ R$.

By the fact that $R^\sim \circ R$ is symmetric, $(z, x) \in R^\sim \circ R$.

From this and euclideaness of R , $(z, x) \in R$, and therefore $(x, z) \in R^\sim$.

Combining this with $(z, y) \in (R \cup R^\sim)^n \circ R$ we get that $(x, y) \in R^\sim \circ (R \cup R^\sim)^n \circ R$.

From this it follows by the **induction hypothesis** that $(x, y) \in R$.

(ii) Assume

$$(x, y) \in R^\sim \circ R^\sim \circ (R \cup R^\sim)^n \circ R.$$

Then there is a z with $(x, z) \in R^\sim$ and $(z, y) \in R^\sim \circ (R \cup R^\sim)^n \circ R$.

From the first of these, $(z, x) \in R$.

From the second of these, by **induction hypothesis**, $(z, y) \in R$.

From $(z, x) \in R$ and $(z, y) \in R$ by euclideaness of R , $(x, y) \in R$.

How to Read Proofs Written by Others

Try to discern the structure.

- What are the givens?
- What is to be proved?
- What are the subproofs?
- How are the definitions used to make explicit what is given?
- In case of an induction proof:
 - Is it induction on the natural numbers or structural induction?
 - What is the base case? Or: What are the base cases?
 - What is the induction hypothesis?
 - How and where is the induction hypothesis used?

Example: Analysis of Proof “Bisimilar states are modally invariant”

- We have available as given the information that we can extract from the definition of bisimulation (invariance, zig and zag conditions).
- We have to prove that **all** modal formulas have a certain property. A good thing to try is proof by induction on the structure of the formula.
- What are the base cases? $\varphi = \top$, $\varphi = p$ (for these are the base cases in the definition of modal formulas).
- What are the inductive cases? $\varphi = \neg\psi$, $\varphi = \psi_1 \vee \psi_2$, $\varphi = \langle a \rangle\psi$ (for these are the recursive cases in the definition of modal formulas).
- What is the induction hypothesis? That ‘simpler’ formulas have

the same truth values in bisimilar states.

- Once we see all this, it is not difficult to **understand** the proof by recognizing its structure.

How to Get From a Definition to a Haskell Implementation

```
module LAI12 where

import List
```

An epistemic model consists of:

- a set of worlds W ,
- a valuation function V ,
- for each agent b a relation R_b on the set of worlds.

We can represent the relations as the set of all triples of the form (b, x, y) , where b is an agent and $(x, y) \in R_b$.

But then we have to know the set of agents.

Thus, we need:

- A set of worlds,
- A set of agents,
- A valuation function,
- A set of triples representing the accessibility relations for the worlds.

We can represent the sets as lists, and the valuation function as a list of pairs (w, Q) , where Q is a list of atomic propositions.

This gives:

- A list of worlds,
- A list of agents,
- A valuation function represented as a list of pairs,
- A list of triples representing the accessibility relations for the agents.

It is useful to distinguish a list of actual worlds.

This gives us:

- A list of worlds,
- A list of agents,
- A valuation function represented as a list of pairs,
- A list of triples representing the accessibility relations for the agents.
- A sublist of the list of worlds representing the actual worlds.

A final decision to be made: what **kind of thing** is a world?

Haskell allows the following answer: a world can be anything. The way to represent **anything** is by means of a **type variable**.

Finally, we arrive at the following datatype:

```
data EpistM world = Mo
    [world]
    [Agent]
    [(world, [Prop])]
    [(Agent, world, world)]
    [world] deriving (Eq, Show)
```

An **instance** of this datatype looks like this:

```
model1 :: EpistM Integer
model1 = Mo worlds agents val rel actual
  where ...
```

Example 2: From a Definition to a Haskell Function

Let A be an ordered set.

Take a function $f : A \rightarrow A$.

Assume f is monotone. This means that, for $a \in A$.

$$a \leq fa \leq ffa \leq fffa \dots$$

The least fixpoint of f on a is $f^n a$ for the least n with $f^n a = f^{n+1} a$.

Generalizing over the argument, this can be defined as:

$$\mathbf{lfp} \ f \ x = \begin{cases} x & \text{if } fx = x \\ \mathbf{lfp} \ f \ (fx) & \text{if } fx \neq x \end{cases}$$

From here to the implementation is trivial:

```
lfp :: Ord a => (a -> a) -> a -> a
lfp f x | x == f x = x
        | otherwise = lfp f (f x)
```

How to Read and Understand a Haskell Function

- What does the type specification mean?
- Is there a definition that the function intends to implement?
- Is it possible to play with the function on simple values, in order to test it out and get familiar with it?

Example: Implementation of Bisimulation Minimal Model

- First grasp the notion of a bisimulation.
- Next, understand why bisimilarity is an equivalence.
- Next, understand how equivalences induce partitions.
- Next, see that mapping worlds to their equivalence classes modulo bisimilarity is a simplification.
- Finally, understand how the partition refinement algorithm achieves this.

Now, and only now, you are ready for the code.

Example: Different Representations of Relations

Relations as characteristic functions.

Appropriate type: $a \rightarrow a \rightarrow \text{Bool}$.

Relations as lists of pairs. Appropriate type $[(a, a)]$.

Back and forth between these representations: first think about the types.

From lists of pairs to characteristic functions. The conversion function has type $[(a,a)] \rightarrow a \rightarrow a \rightarrow \text{Bool}$.

We need an equality type, since we are testing elementhood.

```
list2cf :: Eq a => [(a,a)] -> a -> a -> Bool
list2cf pairs x y = elem (x,y) pairs
```

From characteristic functions to lists of pairs. Now we need a domain.

```
cf2list :: (a -> a -> Bool) -> [a] -> [(a,a)]
cf2list f xs = [ (x,y) | x <- xs, y <- xs, f x y ]
```

Homo Ludens: Play for Understanding

```
LAI12> [(x,y) | x <- [1..3], y <- [1..3] ]
[(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)]
LAI12> list2cf (zip [0..100] [1..]) 5 5
False
LAI12> list2cf (zip [0..100] [1..]) 5 6
True
LAI12> list2cf (zip [0..100] [1..]) 4 5
True
LAI12> cf2list (<=) [1..3]
[(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)]
LAI12> cf2list (>) [1..3]
[(2,1), (3,1), (3,2)]
```

Example: Different Representations of Equivalences

Equivalences as characteristic functions. Type $a \rightarrow a \rightarrow \text{Bool}$.

Partitions corresponding to equivalences. Type $[[a]]$.

Back and forth between the two representations. To get from characteristic functions to partitions we need the domain as well.

```
cf2partition :: (a -> a -> Bool) -> [a] -> [[a]]
cf2partition f [] = []
cf2partition f (x:xs) = xblock : cf2partition f rest
  where (xblock,rest) =
        (x: filter (f x) xs, filter (not.(f x)) xs)
```

Conversely, we just need the partition. But now we need a type with equality, because of the use of `elem`.

```
bl :: Eq a => [[a]] -> a -> [a]
bl part x = head (filter (elem x) part)

partition2cf :: Eq a => [[a]] -> a -> a -> Bool
partition2cf part x y = elem x (bl part y)
```

Homo Ludens Plays Again

```
LAI12> b1 [[1..4],[5..10]] 7
[5,6,7,8,9,10]
LAI12> cf2partition (==) [1..5]
[[1],[2],[3],[4],[5]]
LAI12> partition2cf [[1,2],[3,4]] 1 3
False
LAI12> partition2cf [[1,2],[3,4]] 3 4
True
LAI12> partition2cf [[1,2],[3,4]] 3 3
True
```

Playing with the conversion functions boils down to **testing them out**.

References

- [1] K. Doets and J. van Eijck. *The Haskell Road to Logic, Maths and Programming*, volume 4 of *Texts in Computing*. King's College Publications, London, 2004.