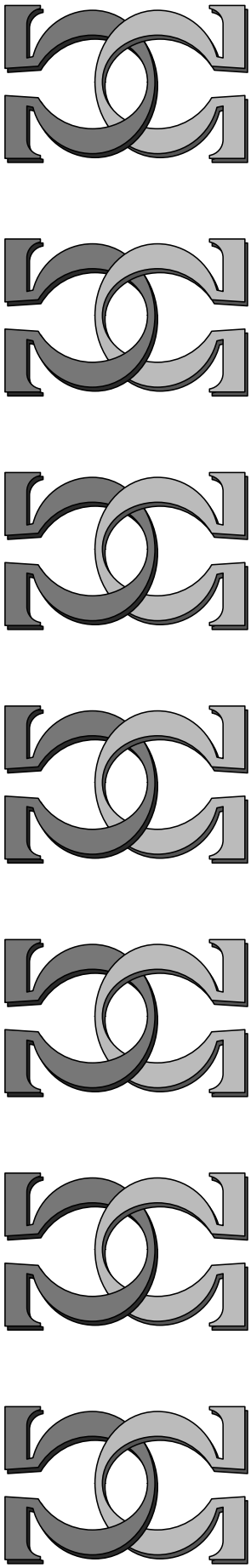

CDMTCS
Research
Report
Series

Formalising Martin-Löf's Theorem Using Coq

D. Britten

University of Auckland, New Zealand

CDMTCS-522
February 2018

Centre for Discrete Mathematics and
Theoretical Computer Science

Formalising Martin-Löf's Theorem Using Coq

by

Daniel Britten

supervised by

Cristian S. Calude

and with guidance from

Monica Marcus

A report submitted as part of the requirements for Computer Science 789 - Honours Dissertation

University of Auckland - 2017-2018

Introduction

A classic question in logic and computer science is whether there exist undecidable mathematical statements. In his 1995 paper [1] Per Martin-Löf proved that, on the intuitionistic conception of the key notions, there are no absolutely undecidable propositions. A key aim of this dissertation was to formalise some of the reasoning from Martin-Löf's work.

The central notion to formalise is: 'can be known'. Since both possibility and knowledge have been modelled using modal logics for many years, a key approach explored was formalising Martin-Löf's argument in a modal logic.

It has turned out to be challenging to formalise Martin-Löf's argument. A chapter is dedicated to unpacking challenges and failed attempts at formalising his theorem. It is still unclear whether an accurate formalisation has been found.

The most promising formalisation involves a 'Constructive S4' modal logic [2]. This formalisation is explored and includes a proof of the Martin-Löf Theorem in this system. Concerns with the system are also discussed.

It remains difficult to fully comprehend the meaning of the Martin-Löf Theorem. But the hope is that this work is some progress towards a fuller understanding of the intriguing result proved by Martin-Löf.

Acknowledgements

I would like to thank Cris Calude for his guidance and support throughout this project. I am very grateful for his advice, encouragement, and understanding. I would also like to thank Monica Marcus for her enthusiastic and extensive input into this project, and especially for her tireless efforts working on finding an adequate formalisation and proof of Martin-Löf's Third Law. Lastly, I would also like to thank Patrick Girard for his assistance with understanding intuitionistic logic more deeply.

Contents

1 The Coq Proof Assistant **6**
- 1.1 Interactively viewing proofs . . . 6
- 1.2 A typical Coq Proof . . . 6
- 1.3 An overview of relevant tactics . . . 7
- 1.4 Confidence in different types of Coq proofs . . . 9

2 The Martin-Löf Theorem **10**
- 2.1 Unpacking the key notions . . . 10
  - 2.1.1 Proposition . . . 10
  - 2.1.2 Truth . . . 10
  - 2.1.3 Falsity . . . 10
  - 2.1.4 Knowledge . . . 11
  - 2.1.5 Possibility . . . 11
- 2.2 Unpacking the Martin-Löf Theorem . . . 11
  - 2.2.1 Objective vs. subjective mathematics . . . 11
  - 2.2.2 Does objective mathematics coincide with subjective mathematics? . . . 11
  - 2.2.3 Objective mathematics vs. subjective mathematics . . . 12
  - 2.2.4 Martin-Löf's argument . . . 12

3 Formalising Modal Logics in Coq **15**
- 3.1 Motivation and background . . . 15
  - 3.1.1 The Benzmuller and Paleo paper . . . 15
  - 3.1.2 Bi-modal logic . . . 15
- 3.2 Coq formalisation of modal logic . . . 15
  - 3.2.1 Defining an embedding of modal logic in Coq . . . 16
  - 3.2.2 Increasing the usability by defining tactics . . . 18
  - 3.2.3 Examples . . . 20
- 3.3 Coq formalisation of bi-modal logic . . . 21
  - 3.3.1 Defining the embedding of bi-modal logic in Coq . . . 21
  - 3.3.2 Increasing usability by defining tactics . . . 21
  - 3.3.3 Examples . . . 22
  - 3.3.4 The Third Law . . . 22

4 Formalising the Martin-Löf Theorem - Challenges and Failed Attempts 24
4.1 A challenge - double negation elimination and the completeness of intuitionistic logic . . . 24
4.1.1 Example: double negation elimination . . . 24
4.1.2 Completeness theorems . . . 26
4.2 Failed attempts . . . 27
4.2.1 A failed Coq formalisation that oversimplifies the notion 'undecidable' 27
4.2.2 A failed Coq formalisation that oversimplifies the notion 'knowable' . 29

5 Formalising the Martin-Löf theorem - Formalisation in CS4+Int 31
5.1 Axioms . . . 31
5.1.1 From CS4 . . . 31
5.1.2 From intuitionistic logic . . . 31
5.2 Proof of Martin-Löf's Third Law . . . 32
5.3 Coq definitions . . . 32
5.3.1 Propositional atoms . . . 32
5.3.2 Modal formula syntax . . . 32
5.3.3 Notation . . . 33
5.3.4 Using Coq's logic as a meta-language . . . 33
5.3.5 Axioms in Coq . . . 33
5.4 Proof of Martin-Löf's Third Law . . . 34
5.4.1 Proof mirroring the structure of Marcus' proof . . . 34
5.4.2 Alternate proof of Martin-Löf's Third Law . . . 34
5.5 Reasonableness of the axioms . . . 35
5.5.1 Considering the axioms themselves . . . 35
5.5.2 Considering some lemmas . . . 35
5.6 Extending to Martin-Löf's Theorem . . . 36

6 Formalising the Martin-Löf Theorem - Formalisation of Intuitionistic Logic 38
6.1 Motivation and background . . . 38
6.2 Defining an embedding of intuitionistic logic . . . 38
6.2.1 Atoms . . . 38
6.2.2 Propositions . . . 38
6.2.3 Proofs . . . 39
6.2.4 Notation . . . 40
6.2.5 Lemmas . . . 40
6.3 Proof of Martin-Löf's Third Law . . . 42
6.4 Proof of the Martin-Löf Theorem . . . 42
6.5 Comments on this formalisation . . . 43

7 Open Questions **44**
7.1 Double negation elimination and the Martin-Löf Theorem 44
7.2 Simplifying the axioms in the CS4 + Int formalisation 44
7.3 Implications of the 'Non-strictly positive occurrence' error in the intuitionistic logic formalisation 45

Chapter 1

The Coq Proof Assistant

This chapter is an introduction to and overview of the proof techniques relating to the Coq Proof assistant [3] that are used in this dissertation.

1.1 Interactively viewing proofs

The entire text of this dissertation is comprised of Coq source files which include all the proofs given. The reader is recommended to download CoqIde from

https://coq.inria.fr/download

and use the Coq source files available from

https://github.com/Coda-Coda/MartinLoefTheorem-Dissertation/releases/tag/cdmtcs

This will allow the reader to step through any of the proofs in this dissertation interactively.

1.2 A typical Coq Proof

Because of the nature of the topic covered by this dissertation, some of the proof techniques used are not entirely standard (for instance `Axioms` are often used) but most of the same elements are still used. Below is an annotated fairly standard Coq proof that highlights some of the ways Coq is used to assist with proofs in this dissertation.

`Inductive` *boolean* :=
| *true*
| *false*.

Above a data type, *boolean*, is defined which can have either of the values listed. Data type definitions of this kind will be used to define propositional or modal formulas as their own type.

`Definition` *not* (*x*:*boolean*) : *boolean* :=

```
match x with
 | true ⇒ false
 | false ⇒ true
end.
```

Above a function *not* is defined with the type signature *boolean* → *boolean*. Very few function definitions of this kind are used in this dissertation.

```
Lemma example_lemma : ∀ (b:boolean), b = not (not b).
Proof.
destruct b.
 - Case b = true      reflexivity.
 - Case b = false     reflexivity.
Qed.
```

Above is an example of a Coq proof. The statement of the proof appears first, followed by the 'proof script' which instructs Coq as to which steps to take to prove the statement. Each step involves using a *tactic* which can be a basic step or be a complex semi-automated step. This dissertation involves many proofs and also defines some *tactic*s in order to aid with proofs.
A Coq proof can be stepped through interactively *tactic* by *tactic*. At each stage Coq will show the remaining `goal` that needs to be proven at that stage.

1.3 An overview of relevant tactics

In general, Coq does not automatically prove statements. Rather, the process is guided by the user inputting tactics which control the flow of the proof. Below are a selection of tactics that are commonly used in proofs in this dissertation.

- `reflexivity` will simplify the goal and then solve a goal of the form a=a.

  ```
  Example reflexivity_example : 1=1.
  Proof.
  reflexivity.
  Qed.
  ```

- `simpl` will simplify a goal.

  ```
  Example simpl_example : 1 + 2 = 3.
  Proof.
  simpl.  The goal now is 3 = 3.
  reflexivity.
  Qed.
  ```

- `intros` will typically introduce quantified variables as new arbitrary variables. Usually used with goals of the form ∀ p q, ...

`Example` *intros_example* : $\forall\ n,\ n + 1 = n + 1$.
`Proof.`
`intros.` The context now contains n : *nat*, and the `goal` is $n + 1 = n + 1$.
`reflexivity.`
`Qed.`

- `exact` will solve a goal that looks exactly like a current hypothesis. In the example below, p is introduced as the hypothesis H and then the goal p can be solved with the tactic `exact` *H*.

  `Example` *exact_example* : $\forall\ p,\ p \to p$.
  `Proof.`
  `intros` *p H*.
  `exact` *H*.
  `Qed.`

- `unfold` will unpack a definition and rewrite the `goal` accordingly.

  `Definition` *double_neg* (*p*:`Prop`) := $\neg\ \neg\ p$.

  `Example` *unfold_example* : $\forall$ *p*:`Prop`, $p \to$ *double_neg p*.
  `Proof.`
  `intro.`
  `unfold` *double_neg*. Now the `goal` is $p \to \neg\ \neg\ p$.
  `firstorder`. Proof automation for first order logic can complete the proof.
  `Qed.`

- `destruct` facilitates analysing different cases.

  `Example` *destruct_example* : $\forall$ *b*: *boolean*, $b =$ *true* $\vee\ b =$ *false*.
  `Proof.`
  `destruct` *b*.
  `left. reflexivity.` Case: $b =$ *true*.
  `right. reflexivity.` Case: $b =$ *false*.
  `Qed.`

- `pose` *proof* can be used to bring an axiom into the context as shown below. Note how *example1* is followed by p and $\neg\ q$, they are then bound to A and B respectively, giving the correct instantiation of the axiom *example1*.

  `Axiom` *example1* : $\forall\ A\ B, \neg\ A \to A \vee B \to B$.

  `Example` *pose_proof_example* : $\forall\ p\ q, \neg\ p \to p \vee \neg\ q \to \neg\ q$.
  `Proof.`
  `intros` *p q*.
  `pose` *proof example1 p* ($\neg\ q$) `as` *example1_axiom*.
  `exact` *example1_axiom*.

`Qed.`

- `apply` will apply a theorem to the current goal, as shown below.

  `Example` *apply_example* : $\forall\ p\ q, \neg\ p \rightarrow p \vee \neg\ q \rightarrow \neg\ q$.
  `Proof.`
  `intros` $p\ q$.
  `apply` *example1*.
  `Qed.`

1.4 Confidence in different types of Coq proofs

It is important to note that different ways of using Coq have important implications for how trustworthy the resulting proofs are and for which sections of code need to be checked to be trustworthy before trusting a particular proof.
When Coq is used without adding any additional assumptions and for its regular usage we can be very confident of the system's consistency and the trustworthiness of proofs. In general, if the Coq definitions used in a theorem are fully understood, the user code leading to a proof of it would not need to be manually checked to be trustworthy.
In contrast, when the code leading to a proof makes use of assumptions then it is important to carefully check that the assumptions do not lead to unintended results. For example, consider the statement below.

`Parameter` i : `Type.`

The `Parameter` keyword is an example of this. In the code above, we assume the existence of i and that it is a `Type`. In this case, this is a safe assumption but it is important to note that the trustworthiness of proofs in a Coq file depends on the trustworthiness of such statements.
For example, the following use of `Parameter` would make Coq's logic inconsistent:
`Parameter` x: *False*.
This assumes the existence of x of type *False*, which essentially assumes the existence of a proof of *False* (which should be impossible).
A second consideration is whether the proof involves an embedding and 'lifted connectives', such as is the case with the Modal Logic formalisation in a later chapter. This technically falls under the idea of being sure that "the Coq definitions used in the theorem are fully understood". Essentially, if an entire logical system has been newly defined, then any theorems proven in such a system are only as trustworthy as the definitions given in the user code.
These considerations are relevant to aspects of the formalisations in this dissertation, and also relevant are the standard considerations outlined on a Coq FAQ site [4] which include trusting the integrity of the system Coq is running on and trusting the theory behind Coq.

Chapter 2

The Martin-Löf Theorem

The Martin-Löf Theorem is introduced in the paper Verificationism Then and Now (1995, 2013)[1, 5]. The conclusion of the paper is that, on the intuitionistic conception, there are no absolutely undecidable propositions.
A large portion of the paper involves carefully unpacking the notions of a proposition, truth, falsity, knowledge and possibility.
The purpose of this dissertation and the ongoing work in parallel to it is to formalise the proof outlined by Martin-Löf in some sound and complete logical system as well as in a proof assistant such as Coq.

2.1 Unpacking the key notions

For a full treatment of the key notions in the Martin-Löf Theorem see the paper Verificationism Then and Now [5]. Here an overview is given.

2.1.1 Proposition

A proposition is defined by its introduction rules. For example, the proposition $A \vee B$ gets its meaning from that the introduction rule for $\vee$ requires either a proof for A or a proof for B in order to have a proof for $A \vee B$.

2.1.2 Truth

A proposition is true if it has a proof.

2.1.3 Falsity

A proposition is false, if there exists a hypothetical proof of $\bot$ from it.

2.1.4 Knowledge

A proposition can be known to be true if there exists a proof for it.

2.1.5 Possibility

The 'if there exists' in the definition of knowledge refers to an intuitionistic understanding of these words. That is, possibility in principle. For example, in order for a statement involving a large number to be known to be true, there need not be a proof detailing every step of that proof - rather it would be sufficient to demonstrate the existence of an algorithm that would, in principle, produce the required step by step proof.

2.2 Unpacking the Martin-Löf Theorem

The sections below describe the informal proof of the Martin-Löf Theorem. This content was kindly provided by Cristian Calude [6], with only minor alterations made by myself.

2.2.1 Objective vs. subjective mathematics

- Objective mathematics consists of the body of mathematical propositions, constructive or not, which hold true in an absolute sense. Peano Arithmetic or Zermelo-Fraenkel set theory are parts of it.
- Subjective mathematics consists of all mathematical truths *humanly demonstrable* (or *provable* or *knowable*), in a constructive manner or not.

2.2.2 Does objective mathematics coincide with subjective mathematics?

Gödel *accepted* Hilbert's rejection of the existence of absolutely unsolvable problems because otherwise,

> "it would mean that human reason is utterly irrational by asking questions it cannot answer, while asserting emphatically that only reason can answer them" [7, p. 324-325]

but he found Turing's argument *inconclusive*:

> "Turing gives an argument which is supposed to show that mental procedures cannot go beyond mechanical procedures. However, this argument is inconclusive. What Turing disregards completely is the fact that mind, in its use, is not static, but constantly developing, i.e., we understand abstract terms more and more precisely as we go on using them ... though at each stage the number and precision of the abstract terms at our disposal may be finite, both ... may converge toward infinity ..." [8, p. 306]

Gödel's answer (Gibbs lecture "Some Basic Theorems on the Foundations of Mathematics and their Implications", [9]) based on his incompleteness theorem is a *disjunctive conclusion*:

> "Either mathematics is incompletable in this sense, that its evident axioms can never be comprised in a finite rule, that is to say, the human mind (even within the realm of pure mathematics) infinitely surpasses the powers of any finite machine, or else there exist absolutely unsolvable diophantine problems of the type specified."

Martin-Löf's answer based on a *constructive* interpretation of the notions of 'true', 'false' and 'can be known' [1]:

> There are no propositions which can neither be known to be true nor be known to be false.

For the non-constructive mathematician:

> No propositions can be effectively produced (i.e. by an algorithm) of which it can be shown that they can neither be proved constructively nor disproved constructively. There may be absolutely unsolvable problems, but one cannot effectively produce one for which one can show that it is unsolvable.

2.2.3 Objective mathematics vs. subjective mathematics

Emil Post writes: [10, 11, p. 200]

> "A fundamental problem is the question of the existence of absolutely undecidable propositions, that is, propositions which in some a priori fashion can be said to have a determined truth-value, and yet cannot be proved or disproved by any valid logic."

We will only require that the objective mathematics contains the subjective mathematics.

Furthermore, in contrast with Feferman [12], *we will include in subjective mathematics all statements provable by any methods, axiomatic (dynamic, not only static), constructive, computational or by methods currently not yet discovered.*

2.2.4 Martin-Löf's argument

Constructive logical interpretations:

- The proposition A can be known to be true if we have a proof for A.
- The proposition $A \vee B$ can be known to be true if we have a proof for A or we have a proof for B.

- The proposition $A \wedge B$ can be known to be true if we have a proof for A and we have a proof for B.
- The proposition $A \to B$ can be known to be true if we have an algorithm which converts any proof for A into a proof for B.
- The proposition $\sim A$ can be known to be true if we have a proof for $A \to (0 = 1)$.
- The proposition A *can be known to be false* if we have a proof for $\sim A$.
- The proposition A *cannot be known to be true* if we have an algorithm which tests and rejects any given 'proof' which purports to demonstrate A.
- If the proposition A can be known to be true, then A is true.
- Martin-Löf's notions of *can be known to be true/false* are not related to any fixed formal system.

Fact 1. [Unknowability of truth entails knowability of falsity] *If the proposition A cannot be known to be true, then A can be known to be false.*

Proof: To prove that A can be known to be false we have to show that $\sim A$, i.e. $A \to (0 = 1)$ can be known to be true. To this aim we need an algorithm $\mathcal{B}$ to convert any proof of A into a proof of $(0 = 1)$. The algorithm $\mathcal{B}$ returns anything output by the algorithm $\mathcal{A}$ provided by the hypothesis, i.e. noting: vacuously, the implication holds.

Comment: The proof constructively produces positive information from negative information.

Fact 2. *If A can be known to be true and B can be known to be true, then $A \wedge B$ can be known to be true.*

Fact 3. [Absolute consistency] *The proposition $(0 = 1)$ cannot to be known to be true.*

Proof: The proposition $\sim(0 = 1)$ can be known to be true because $(0 = 1) \to (0 = 1)$ is provable using the identity algorithm, so $(0 = 1)$ can be known to be false, i.e. it is false. No proof can demonstrate $(0 = 1)$ because otherwise it would be true: the algorithm rejects any proof candidate.

Fact 4. [Law of contradiction] *One and the same proposition A cannot both be known to be true and be known to be false.*

Proof: By hypothesis we have a proof demonstrating A and a proof demonstrating $\sim A$, i.e. $A \to (0 = 1)$. Then we can demonstrate $(0 = 1)$, contradicting Fact 3.

Fact 5. [Law of excluded middle] *There is no proposition which can neither be known to be true nor be known to be false, i.e. there is no absolutely unprovable proposition.*

Proof: If A is a proposition which cannot be known to be true, then by Fact 1, A can be known to be false, a contradiction.

Chapter 3

Formalising Modal Logics in Coq

3.1 Motivation and background

The Martin-Löf Theorem includes the notion of knowledge as well as the notion of possibility in key ways. Both possibility and knowledge have been modelled using modal logics for many years, and so it seemed likely that incorporating or adapting some of those techniques would be useful in formalising the Martin-Löf theorem and its proof.

3.1.1 The Benzmuller and Paleo paper

The paper 'Interacting with Modal Logics in the Coq Proof Assistant' [13] sets out a useful way to embed modal logics in the Coq proof assistant. The approach they put forward in their paper is also extensible and so had the potential to be adapted as necessary when formalising the Martin-Löf Theorem. Their work was very helpful.
In their paper they state: *"the main contribution described in this paper - convenient techniques for leveraging a powerful proof assistant such as Coq for interactive reasoning for modal logics - may serve as a starting point for many interesting projects."* [13, p. 2].

3.1.2 Bi-modal logic

One of the approaches was to formalise the theorem in a bi-modal logic, representing knowability and possibility as modal operators. This approach is shown below, based almost entirely on the Coq code from the Benzmuller paper, with some minor adaptations for it to be bi-modal logic.

3.2 Coq formalisation of modal logic

Here the approach to formalising $K5$ given in the Benzmuller and Paleo paper is outlined and explored with reference to exercises and examples from First-Order Modal Logic (Fitting and Mendelsohn, 1998)[14]. The purpose of this section is to explore the Benzmuller and

Paleo approach when applied to standard modal logics before extending it to handle bi-modal logic.

3.2.1 Defining an embedding of modal logic in Coq

The Coq code in this subsection is directly from the Benzmuller and Paleo Paper [13], the comments have been added.
It is worth noting that to check the trustworthiness of any theorem using the techniques from the Benzmuller and Paleo paper, it is necessary to check the trustworthiness of the user code defining the modal logic embedding (below), and especially the impact of any adaptations to their original technique such as the adaptation to bi-modal logic later in this chapter.

Worlds, objects, modal propositions and the accessibility relation

`Parameter` i: `Type`. Here we declare/assume a type i as the type for worlds.

`Parameter` u: `Type`. Here we declare/assume a type u as the type for objects/individuals.

`Definition` $o := i \to$ `Prop`. Here we define o to be the type which takes a world and gives a meta-language proposition. o then, is the type of modal propositions, which when given a world are then a meta-language proposition.
The intended usage is to have a modal formula, say p (with type o), with the meaning of $(p\ w)$ to be that the modal proposition represented by p is true at world w (where w is a world, with type i).
The meanings of the definitions of i, u and o are fixed further by their usage in the next definitions. It is also worth noting that the reason why undescriptive names are used is because i, u and o will need to be written often and it would be inconvenient if their names were long.

`Parameter` r: $i \to i \to$ `Prop`. Here we declare/assume a function/predicate of arity 2 defined over worlds, to be the accessibility relation for worlds.

Connectives

Here the lifted connectives are defined. Each lifted connective (except *mnot*) takes two modal propositions (with type o) and a world (with type i) and gives a 'meta-language' proposition (with the standard Coq type `Prop`) reflecting the meaning of the connective.

`Definition` *mnot* $(p\colon o)(w\colon i) := \neg\ (p\ w)$.
`Definition` *mand* $(p\ q{:}o)(w\colon i) := (p\ w) \wedge (q\ w)$.
`Definition` *mor* $(p\ q{:}o)(w\colon i) := (p\ w) \vee (q\ w)$.
`Definition` *mimplies* $(p\ q{:}o)(w{:}i) := (p\ w) \to (q\ w)$.
`Definition` *mequiv* $(p\ q{:}o)(w{:}i) := (p\ w) \leftrightarrow (q\ w)$.
`Definition` *mequal* $(x\ y\colon u)(w\colon i) := x = y$.

Here we define a more natural prefix and infix notation for the lifted connectives.

Notation "$m\neg\ p$" := (*mnot p*) (at level 74, right associativity).
Notation "$p\ m\wedge\ q$" := (*mand p q*) (at level 79, right associativity).
Notation "$p\ m\vee\ q$" := (*mor p q*) (at level 79, right associativity).
Notation "$p\ m\rightarrow\ q$" := (*mimplies p q*) (at level 99, right associativity).
Notation "$p\ m\leftrightarrow\ q$" := (*mequiv p q*) (at level 99, right associativity).
Notation "$x\ m=\ y$" := (*mequal x y*) (at level 99, right associativity).
Note that the notation for each of the connectives in the embedded modal logic is an 'm' followed by the connective. So there is $m\neg$ and $m\vee$ for example. Also, below *mforall* and *mexists* are defined.

Quantifiers

Here the lifted versions of the quantifiers are defined. The quantifiers take a type, a function/predicate which takes elements of that type and gives a modal proposition, and a world, and gives a 'meta-language' proposition' (with type Prop) reflecting the meaning of the quantifier.

Definition A {t: Type}(p: $t \rightarrow o$)(w: i) := $\forall\ x,\ p\ x\ w$.
Definition E {t: Type}(p: $t \rightarrow o$)(w: i) := $\exists\ x,\ p\ x\ w$.

If a world is not given, then $A\ t\ p$ will have type $i \rightarrow$ Prop and so it will be a modal proposition itself. This will allow the expression of modal formulae such as $\forall x, P(x) \rightarrow P(x)$. In this formula P would correspond to p in the definitions above.

Here more natural notations for A and E are defined. The definitions for A and E make use of Coq's type inference ability so t doesn't need to be explicitly entered (in many cases t would be the type for objects/individuals u), it is inferred from the type of p.

Notation "'mforall' x , p" := (A (fun $x \Rightarrow p$))
(at level 200, x *ident*, right associativity) : *type_scope*.
Notation "'mforall' x : t , p" := (A (fun x:$t \Rightarrow p$))
(at level 200, x *ident*, right associativity,
format "'[' 'mforall' '/ ' x : t , '/ ' p ']'") : *type_scope*.
Notation "'mexists' x , p" := (E (fun $x \Rightarrow p$))
(at level 200, x *ident*, right associativity) : *type_scope*.
Notation "'mexists' x : t , p" := (E (fun x:$t \Rightarrow p$))
(at level 200, x *ident*, right associativity,
format "'[' 'mexists' '/ ' x : t , '/ ' p ']'") : *type_scope*.

Modalities

Definition $\Box$ (p: o) := fun $w \Rightarrow \forall\ w1,\ (r\ w\ w1) \rightarrow (p\ w1)$.
Definition $\Diamond$ (p: o) := fun $w \Rightarrow \exists\ w1,\ (r\ w\ w1) \wedge (p\ w1)$.

$\Box$ is defined as a function which takes a modal proposition and a world, and returns a 'meta-level' proposition stating that for all worlds related to that world the modal proposition is true at those worlds.

$\Diamond$ is defined similarly as a function which takes a modal proposition and a world, and returns a 'meta-level' proposition stating that there exists a world related to that world, where the modal proposition is true.
Since the type of both $\Diamond$ and $\Box$ when applied to a modal proposition, is $i \rightarrow$ `Prop`, then $\Box$ p and $\Diamond$ p are of type o (the type of modal propositions).

Validity

`Definition` V (p: o) := $\forall$ w, p w. Here the definition of validity for a modal formula is given. V is defined as a function/predicate that takes a modal proposition and gives a 'meta-level' proposition stating that the given modal proposition is true at all worlds. So for any modal proposition p, at the meta-level V p is true iff p is valid.

`Notation "[ p ]" :=` (V p). Here a notation is defined so that we can write "p is valid" as $[p]$ rather than (V p).

The definitions required to embed modal logic are now all defined. The tactics defined below only add to the usability of the above definitions and do not alter what propositions are expressible and provable in this embedding of modal logic in Coq like each of the definitions above have.

3.2.2 Increasing the usability by defining tactics

The tactics defined below allow proofs to be carried out without dealing directly with the intricate aspects of the definitions of the embedding. Instead, the familiar technique of working with introduction and elimination rules can be used. Only some new tactics are needed, as the standard Coq tactics work fine with many of the defined notions. Below a brief example of each new tactic is given; for a comprehensive explanation of these definitions see section 3 of the Benzmuller and Paleo paper [13, p. 5].

Modal validity

`Ltac` mv := `match goal with [|- (`V _`)] => intro end.`

Here the mv tactic is defined such that when the proof state requires proving that a modal formula is valid, applying mv will introduce a new arbitrary world and the goal of the proof state will then be to prove that the modal formula is true at the new world.

`Example` mv_eg : [$mforall$ p, $\Box$ p $m\rightarrow$ p].
`Proof.`
mv.

The proof state goal is now: ($mforallp$:o, $\Box$ p $m\rightarrow$ p) w
`Abort.`

Box introduction

`Ltac` *box_i* := `let` *w* := `fresh` "w" `in let` *R* := `fresh` "R"
`in` (`intro` *w* `at` *top*; `intro` *R* `at` *top*).

`Example` *box_i_eg* : [*mforall p*, □ *p*].
`Proof.`
mv.
`intro.` Here the standard Coq tactic `intro` is sufficient for handling *mforall*.

box_i.
The goal is now: (*x w0*) with *r w w0* as one of the assumptions, where both *w* and *w0* are arbitrary.

`Abort.`

Box elimination

`Ltac` *box_elim H w1 H1* := `match` *type of H* `with`
((□ *?p*) *?w*) ⇒ `cut` (*p w1*);
[`intros` *H1* | (`apply` (*H w1*); `try assumption`)] `end.` *box_elim* is an auxiliary tactic for *box_e*, not intended to be directly called by the user.

`Ltac` *box_e H H1*:= `match goal with` | [⊢ (_ *?w*)] ⇒ *box_elim H w H1* `end.`

`Example` *box_e_eg* : ∀ *w1 w2 p*, *r w1 w2* → (□ *p*) *w1* → *p w2.*
`Proof.`
`intros.`
box_e H0 H1.
The call of *box_e* adds *p w2* to the list of hypothesis, and the requirement *r w1 w2* is automatically solved.

`exact` *H1.*
`Qed.`

Diamond introduction

`Ltac` *dia_i w* := (∃ *w*; `split`; [`assumption` | `idtac`]).

`Example` *dia_i_eg* : ∀ *w1 w2*, ∀ *p*:*o*, *r w1 w2* → *p w2* → (◇ *p*) *w1.*
`Proof.`
`intros.`
dia_i w2.
The call of *dia_i* transforms the goal from ◇ *p w1* to *p w2*, automatically checking that *w2* is reachable from *w1*.

`exact` *H0.*
`Qed.`

Diamond elimination

```
Ltac dia_e H := let w := fresh "w" in let R := fresh "R" in
(destruct H as [w [R H]]; move w at top; move R at top).

Example dia_e_eg : ∀ p w, (◇ p) w → ∃ w0, p w0.
Proof.
intros.
dia_e H.
```

The call of *dia_e* introduces a new arbitrary world *w*, and another arbitrary world *w0* accessible from *w* where *p* is true.

```
∃ w0.
exact H.
Qed.
```

3.2.3 Examples

With the embedding as well as useful tactics defined, standard modal logic lemmas can now be proven here.

```
Lemma example1 : ∀ p q, [□ ( p m∧ q ) m↔ (□ p m∧ □ q)].
Proof.
intros.
mv.
split.
  - intros.
    split.
      + box_i.
        box_e H H1.
        destruct H1.
        exact H0.
      + Similarly, box_i. box_e H H1. destruct H1. exact H1.
  - split.
      + destruct H. box_e H H2. exact H2.
      + Similarly, destruct H. box_e H1 H2. exact H2.
Qed.


Lemma example2 : [mforall p, ◇ p m→ (m¬ (□ (m¬ p)))].
Proof.
mv.
intro.
intro.
dia_e H.
intro.
```

assert(($m\neg$ x) $w0$).
box_e H0 H1.
exact *H1.*
contradiction.
Qed.

3.3 Coq formalisation of bi-modal logic

The formalisation given below relates to an early unsuccessful version of our attempt to formalise the Martin-Löf Theorem.

The modal logic described in the Benzmuller and Paleo paper was adapted to be a bi-modal logic in line with the intention to formalise the Martin-Löf theorem.

3.3.1 Defining the embedding of bi-modal logic in Coq

All the definitions from the Benzmuller and Paleo modal logic embedding [13] in the previous section are used, in addition to new definitions which extend it to a bi-modal logic.

The following definitions are added to extend the logic to be bi-modal. The definitions for K and B mirror $\Box$ and $\Diamond$ respectively.

Parameter *rkb* : $i \to i \to$ Prop. Similarly to with r, here we declare/assume a function/predicate of arity 2 defined over worlds, to be the accessibility relation for for worlds, for knowledge and belief. For bi-modal logic, r is the accessibility relation just for $\Box$ and $\Diamond$.

Definition K (p: o) := fun $w \Rightarrow \forall$ *w1*, (*rkb w w1*) $\to$ (p *w1*).
Definition B (p: o) := fun $w \Rightarrow \exists$ *w1*, (*rkb w w1*) $\wedge$ (p *w1*).

Axioms about the accessibility relation are also added to match definitions in the model.
Axiom reflexivity: $\forall$ w, r w w.
Axiom *reflexivitykb*: $\forall$ w, *rkb w w*.
Axiom *RksubsetRbox*: $\forall$ *w1 w2*, *rkb w1 w2* $\to$ r *w1 w2*.
Axiom *accurate_reasoner*: $\forall$ w, $\forall$ p:o, B p $w \to p$ w.

As before, only the definitions above affect what propositions can be expressed and proven in this embedding of a bi-modal logic. The tactics below only increase the usability.

3.3.2 Increasing usability by defining tactics

The tactics defined here mirror those defined for $\Box$ and $\Diamond$ and would be used similarly.

Ltac *K_i* := let w := fresh "w" in let R := fresh "R"
in (intro w at *top*; intro R at *top*).

```
Ltac K_elim H w1 H1 := match type of H with
((K ?p) ?w) ⇒ cut (p w1);
[intros H1 | (apply (H w1); try assumption)] end.
Ltac K_e H H1:= match goal with | [ ⊢ (_ ?w) ] ⇒ K_elim H w H1 end.

Ltac B_e H := let w := fresh "w" in let R := fresh "R" in
(destruct H as [w [R H]]; move w at top; move R at top).

Ltac B_i w := (∃ w; split; [assumption | idtac]).
```

3.3.3 Examples

Lemma *example1* : $\forall\ p, [K\ p\ m{\rightarrow}\ p]$. i.e. $K\ p \rightarrow p$ is valid.

```
Proof.
intro.
mv.
intro.
K_e H H1.
exact H1.
apply reflexivitykb.
Qed.
```

Lemma *example2* : $\forall\ p\ w1, (B\ p)\ w1 \rightarrow \exists\ w2, (\Diamond\ p)\ w2$. i.e. if $B\ p$ is true at some world then there exists a world where $\Diamond\ p$ is true.

```
Proof.
intros.
B_e H.
apply RksubsetRbox in R.
∃ w1.
dia_i w.
exact H.
Qed.
```

3.3.4 The Third Law

The Third Law of the Martin-Löf theorem can be expressed in this bi-modal logic as below. An important consideration is whether the definitions above capture the meaning of the theorem, in particular whether the formally defined meanings of $\Diamond$ and K when used together capture the notion "can be known".
Informally, Martin-Löf's Third Law states *"From the unknowability of the truth of a proposition, its falsity may be inferred"* [5, p. 12]. This can potentially be captured formally as given below.
Tentative `Theorem` *Third_Law* : $[mforall\ A, m\neg\ (\Diamond\ (K\ A))\ m{\rightarrow}\ (\Diamond\ (K\ (m\neg\ A)))]$.
`Proof.`

`Abort`. A proof is not given here, further work may resolve whether this is provable.

Chapter 4

Formalising the Martin-Löf Theorem - Challenges and Failed Attempts

4.1 A challenge - double negation elimination and the completeness of intuitionistic logic

The completeness theorems for intuitionistic logic can easily be thought to imply that for every intuitionistic formula, there either exists a proof of it, or a proof of its negation. This is what one might expect from the notion of completeness for classical propositional logic, for example, but this is not something that the completeness theorems for intuitionistic logic imply.
We can best demonstrate this by noting that neither $(\neg\neg q \to q)$ nor $\neg(\neg\neg q \to q)$ are intuitionistically provable. Thus there is a formula for which neither it nor its negation is intuitionistically provable.
Below, further details are given to show this is the case, and then the completeness theorems for intuitionistic logic are discussed in relation to this. We find that the intuitionistic completeness theorems do not mean that there don't exist formulae with neither a proof of themselves nor of their negation, so in some sense these theorems are weaker than their classical counterparts.

4.1.1 Example: double negation elimination

$(\neg\neg q \to q)$ is not intuitionistically provable

This we know because of an understanding of the definitions of the inference rules of intuitionistic logic, or by formal analysis of what is provable using some semantics. Essentially, if it were intuitionistically provable, then intuitionistic logic would be no different from classical logic in what is provable. Double negation elimination is equivalent (both classically and intuitionistically) to the law of the excluded middle, which is commonly pointed to as specifically something that is not intuitionistically provable and which separates intuitionistic logic

from classical logic.
Because of such reasons, it is clear that $(\neg \neg q \rightarrow q)$ is not intuitionistically provable.

$\neg$ ($\neg \neg q \rightarrow q$) is not intuitionistically provable

There are a number of ways to show this, a few are given here.

(1) *Because if it were, then $\bot$ would be provable:*
For example, below is a Coq proof of $(\neg (\neg \neg q \rightarrow q)) \rightarrow \bot$.

`Parameter` *q*:`Prop.` Let q be an arbitrary proposition.
`Lemma` *double_neg_example* : $(\neg (\neg \neg q \rightarrow q)) \rightarrow \bot$.
`Proof. firstorder. Qed.` Coq is able to automatically verify this lemma. Note that Coq's core logic which is being used here is intuitionistic.

We can do the same proof more manually using Coq as shown below.
`Lemma` *double_neg_example'* : $(\neg (\neg \neg q \rightarrow q)) \rightarrow \bot$.
`Proof.`
`unfold` *not.*
`intros. apply` *H.*
`intros.` *exfalso.* `apply` *H0.*
`intros. apply` *H.*
`intros. apply` *H1.*
`Qed.`

(2) *As an alternative, here is a natural deduction-style proof of the same result:*
The natural deduction-style proof below mimics the reasoning of the Coq proof above. It makes use only of intuitionistically valid inference rules.

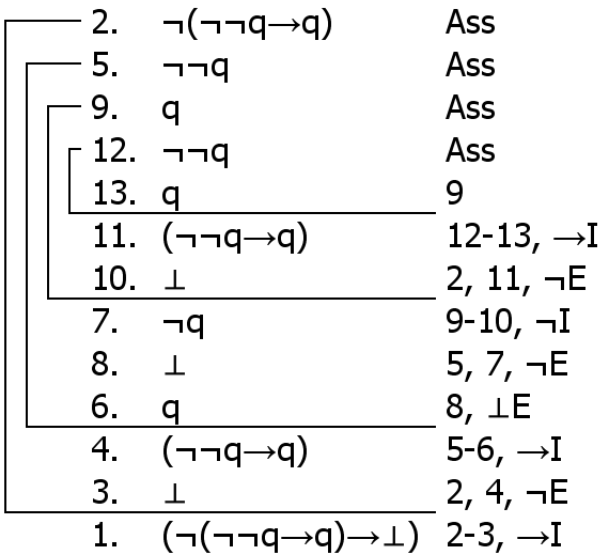

Figure 4.1: NJ proof, created using the Natural Deduction Planner by Declan Thompson [15].

The above three proofs show that $\neg\ \neg\ (\ \neg\ \neg\ q \rightarrow q)$ is intuitionistically provable. This implies that $\neg\ (\ \neg\ \neg\ q \rightarrow q)$ is not intuitionistically provable because if it were, we would clearly then have a proof for $\bot$ which is not possible. Thus $\neg\ (\ \neg\ \neg\ q \rightarrow q)$ is not intuitionistically provable.

(3) *Other results which show that double negation elimination is unprovable intuitionistically:*

We can also see this is the case using the fact that every intuitionistic tautology is a classical tautology, and so if $\neg\ (\ \neg\ \neg\ q \rightarrow q)$ was intuitionistically provable then it would be classically provable, but it is classically false. Thus $\neg\ (\ \neg\ \neg\ q \rightarrow q)$ is not intuitionistically provable. One additional way, would be to use Glivenko's Theorem [16], using the fact that p is classically true iff $\neg\ \neg\ p$ is intuitionistically provable. Thus $\neg\ \neg\ (\ \neg\ \neg\ q \rightarrow q)$ is intuitionistically provable, because $(\ \neg\ \neg\ q \rightarrow q)$ is a classical tautology. So as before, $\neg\ (\ \neg\ \neg\ q \rightarrow q)$ is not intuitionistically provable.

4.1.2 Completeness theorems

Completeness is a notion which in some sense does differ in meaning between classical [17, 18, 19, p. 46] and intuitionistic [16, 19, p. 171] contexts, although it also retains many aspects of its meaning. One example of a difference is that in classical logic if some system is complete, then we can conclude that for every formula either it or its negation is provable. This however, is not the definition of completeness, but rather a consequence of it in classical contexts. For instance, in classical propositional logic, the fact that it is complete is generally defined to mean that every formula that is valid semantically is provable. For

classical logic this then means that if we take an arbitrary formula, we can show that either it or its negation must be provable as follows, the key being the fact that $p \vee \neg p$ is valid in classical logic for all propositions.

Consider classical propositional logic, with semantics and provability (or deduction) defined such as in the Stanford Encyclopedia of Philosophy article on classical logic [20].
Here, if a formula p is valid semantically this is denoted by $\vDash p$. If a formula q is provable this is denoted $\vdash q$.

Classical propositional logic is complete, so every valid formula is provable, if $\vDash p$ then $\vdash p$.

1. Let p be an arbitrary propositional formula.
2. $p \vee \neg p$ is valid, $\vDash (p \vee \neg p)$
3. Thus, by the definition of satisfaction, $\vDash p$ or $\vDash \neg p$
4. Suppose $\vDash p$
    a. Then, by completeness we have that $\vdash p$ so p is provable. We are done. (Either p or $\neg p$ is provable.)
5. Suppose $\vDash \neg p$
    a. Then, by completeness we have that $\vdash \neg p$ so $\neg p$ is provable. We are done. (Either p or $\neg p$ is provable.)
6. Therefore either p or $\neg p$ is provable.

We see here that completeness for classical logic implies that for every p, either p or $\neg p$ is provable. For intuitionistic logic however, since we relied on $p \vee \neg p$ as being valid in step 2 the above reasoning would not work since $p \vee \neg p$ is not valid in intuitionistic logic.
Clarifying this was helpful in the process of gaining a greater understanding of the key notions of the Martin-Löf Theorem.

4.2 Failed attempts

The Martin-Löf theorem considers all notions with regards to "the intuitionistic conception" [5, p. 4] and so it seems plausible that formalising the theorem in a way which works solely with intuitionistic proofs might best capture the ideas in Martin-Löf's proof. Coq's core logic itself is intuitionistic and embodies many of the notions expressed in relation to the Martin-Löf theorem. This section explores two failed attempts at using Coq's default intuitionistic logic to formalise key notions of the Martin-Löf Theorem.

4.2.1 A failed Coq formalisation that oversimplifies the notion 'undecidable'

In Martin-Löf's paper [5], the conclusion is that there are no absolutely undecidable propositions, on the intuitionistic interpretation. A key step in formalising the theorem is to adequately represent the statement of the conclusion formally.

A possible definition of 'absolutely undecidable':

An absolutely undecidable proposition has no proof or disproof. This appears to be the most natural way to define 'absolutely undecidable', but it is important to bear in mind the proposition $\neg\neg q \to q$ as discussed in section 4.1. In Coq this definition could be formalised as follows:

`Definition` *absolutely_undecidable* (p : `Prop`) : `Prop` := $\neg$ (($\exists$ (pf : p), *True*) $\vee$ ($\exists$ (pf : $\neg$ p), *True*)).

An immediate issue with this formalisation of the notion 'absolutely undecidable' is that in Coq (by default) the negation is intuitionistic. As a result the definition above does not correctly capture the meaning of 'no' in the notion 'no proof or disproof'.
Another possible issue is that the definition is very similar to $\forall\ p, \neg\ (p \vee \neg\ p)$ which seems to oversimplify the notion it is attempting to capture. This is shown by the lemmas below. First we proof that there exists a proof of p if and only if p is true.

```
Lemma proof_exists_iff_true : ∀ p:Prop, (∃ pf:p, True) ↔ p.
Proof.
split.
 - intros. destruct H. exact x.
 - intros. ∃ H. apply I.
Qed.
```

Now we can show that *absolutely_undecidable* p is equivalent to $\neg\ (p \vee \neg\ p)$. This is not necessarily a problem, as it may only be showing that *absolutely_undecidable* p is false (since $\neg\ (p \vee \neg\ p)$ is false). But of concern is the simplicity of the proof, which highlights that *absolutely_undecidable* p may have a similar *meaning* to $\neg\ (p \vee \neg\ p)$. This would seem to be an oversimplification of the intended meaning of the notion 'absolutely undecidable'.

```
Lemma absolutely_undecidable_is_oversimplified :
  ∀ p, absolutely_undecidable p ↔ ¬ (p ∨ ¬ p).
Proof.
intros.
unfold absolutely_undecidable.
rewrite! proof_exists_iff_true.
split. intros. exact H. intros. exact H.
Qed.
```

We can define here the conclusion of the Martin-Löf Theorem: that there are no absolutely undecidable propositions. However, as shown below, in this formalisation the statement is equivalent to $\forall\ p, \neg\ (\neg\ (p \vee \neg\ p))$ by a fairly simple proof. This, again, appears to potentially hint that the notion it is trying to formalise has been oversimplified. The meaning of the statement of the Martin-Löf Theorem in this formalisation would be just the meaning of $\forall\ p, \neg\ (\neg\ (p \vee \neg\ p))$ or close to it.

`Definition` *ML_theorem* := $\forall\ p, \neg$ (*absolutely_undecidable* p).

`Lemma` *ML_theorem_equivalence* : *ML_theorem* $\leftrightarrow \forall\ p, \neg\ (\ \neg\ (p \vee \neg\ p))$.
`Proof.`
`unfold` *ML_theorem*.
`split.`
`- intros. pose` *proof H p*. `rewrite` *absolutely_undecidable_is_oversimplified* `in` *H0*. `exact` *H0*.
`- intros. rewrite` *absolutely_undecidable_is_oversimplified*. `apply` *H*.
`Qed.`

Further work would be needed to confirm that this oversimplification is definitely an issue, but it does seem to be problematic.

4.2.2 A failed Coq formalisation that oversimplifies the notion 'knowable'

Here a formalisation is given of the key notion of knowability. This is then extended to a formalisation of Martin-Löf's Third Law. The key issue is that the meaning of the Third Law in this formalisation is closest to "if a proposition is false then that proposition is false" rather than it's expected meaning: "if a proposition cannot be known to be true then it can be known to be false." [5, p. 10]

Definitions

In Coq and in intuitionistic logic more generally, a proposition is true, by definition, iff there exists a proof for it. The precise meaning of "exists" can be debated, but the general idea remains. So a Coq lemma of the form $\forall\ p\ q,\ p \leftrightarrow q$ can be taken to mean for all propositions p is true iff q is true, or it can be taken to mean p has a proof iff q has a proof. (Or a mixture: p is true iff q has a proof).
With this in mind, we can define knowability, K, as follows

`Definition` $K\ (p$: `Prop`$)$: `Prop` $:= p$. Meaning, $K(p) =$ 'p has a proof.'
This definition of K means $\neg\ K(p) =$ '$(p \rightarrow \bot)$ has a proof' as shown in the lemma below.

`Lemma` *not_Kp* : $\forall\ p, (\ \neg\ K\ p) = (p \rightarrow \bot)$.
`Proof. firstorder.` Trivially. `Qed.`

That $p \rightarrow \bot$ has a proof means that we have an algorithm which takes a proof of p and gives a proof of $\bot$. So, $\neg\ K(p)$ means there is an algorithm which takes a proof of p and gives a proof of $\bot$.
It is important to note that $\neg\ K(p) \neq$ 'p has no proof', because, as mentioned before, it is possible for a formula to have no proof and also its negation to have no proof (e.g. $\neg\ \neg\ q \rightarrow q$).
Martin-Löf's Third Law can be formalised using these definitions, however issues arise as discussed below. The Third Law states that "if a proposition cannot be known to be true then it can be known to be false." [5, p. 10]

```
Definition ML_Third_Law := ∀ p, ( ¬ K p) → K ( ¬ p).
Lemma problem_with_MLThirdLaw : ML_Third_Law = ∀ p, ( ¬ p → ¬ p).
Proof.
unfold ML_Third_Law.
unfold K.
reflexivity.
Qed.
```

The above lemma shows that the *ML_Third_Law* statement is in fact the *same* proposition as $\forall\ p, (\neg\ p \rightarrow \neg\ p\ )$ (not merely equivalent to it). When the definition of K is unpacked the statement of the theorem becomes exactly $\forall\ p, (\neg\ p \rightarrow \neg\ p\ )$. This shows that the meaning of the *ML_Third_Law* in this formalisation is that if there exists a proof for $\neg\ p$ then there exists a proof for $\neg\ p$. This appears to be much weaker than the intended meaning: "if a proposition cannot be known to be true then it can be known to be false." [5, p. 10]

Chapter 5

Formalising the Martin-Löf theorem - Formalisation in CS4+Int

Here a potentially adequate formalisation of the Martin-Löf Theorem is explored, a formalisation in the axiom system of CS4 + intuitionistic logic.
CS4 is 'Constructive S4' discussed in Alechina et al (2001)[2]. Here we consider $\Diamond$ to mean 'can be known' or 'is knowable'. Only one axiom of CS4 is required in the proof of the Martin-Löf Theorem. The axiom required is $A \rightarrow \Diamond\ A$. Informally: if a formula is true then it is knowable.

5.1 Axioms

5.1.1 From CS4

$\Diamond\ T : A \rightarrow \Diamond\ A$

5.1.2 From intuitionistic logic

A1 from Introduction to Intuitionistic Logic [21, p. 18]:

$$((A \Rightarrow B) \Rightarrow ((B \Rightarrow C) \Rightarrow (A \Rightarrow C)))$$

Property 11 from Introduction to Intuitionistic Logic [21, p. 24]:

$$((A \Rightarrow B) \Rightarrow (\neg\ B \Rightarrow \neg\ A))$$

Modus Ponens:

$$\text{if } (A \Rightarrow B) \text{ and } A \text{ are true, then } B \text{ is true}$$

5.2 Proof of Martin-Löf's Third Law

The proof below is by Monica Marcus, (personal communication, October 10, 2017).

1. $(\neg \Diamond p \Rightarrow \neg p) \Rightarrow ((\neg p \Rightarrow \Diamond (\neg p)) \Rightarrow (\neg \Diamond p \Rightarrow \Diamond (\neg p)))$ (A1)
2. $p \Rightarrow \Diamond p$ ($\Diamond$ T CS4 axiom)
3. $(p \Rightarrow \Diamond p) \Rightarrow (\neg \Diamond p \Rightarrow \neg p)$ (property 11)
4. $\neg \Diamond p \Rightarrow \neg p$ (Modus Ponens, 2, 3)
5. $(\neg p \Rightarrow \Diamond (\neg p)) \Rightarrow (\neg \Diamond p \Rightarrow \Diamond (\neg p))$ (Modus Ponens 4,1)
6. $\neg p \Rightarrow \Diamond (\neg p)$ ($\Diamond$ T CS4 axiom)
7. $\neg \Diamond p \Rightarrow \Diamond (\neg p)$ (Modus Ponens 6,7)

This proves the formula $\neg \Diamond p \Rightarrow \Diamond (\neg p)$ in CS4 plus a propositional intuitionistic axiom system.

5.3 Coq definitions

The next sections form a Coq version of the above proof. The Coq proof assistant checks that the definitions are used correctly and that the steps in the proof (*tactic*s) are correct, giving an even greater level of rigour.

First it is necessary to define propositional formulas, which can then be reasoned about.

5.3.1 Propositional atoms

`Inductive` *Atoms* :=
| a : *Atoms*
| S : *Atoms* → *Atoms*.

This definition of the type *Atoms* is a straightforward way to capture the notion of there being infinitely many propositional atoms. Here a, $S\ a$, $S\ (S\ a)$, ... are all propositional atoms.

5.3.2 Modal formula syntax

`Inductive` *modal_formula* :=
| *atom* : *Atoms* → *modal_formula*
| *and* : *modal_formula* → *modal_formula* → *modal_formula*
| *or* : *modal_formula* → *modal_formula* → *modal_formula*
| *implies* : *modal_formula* → *modal_formula* → *modal_formula*
| *falsum* : *modal_formula*
| *box* : *modal_formula* → *modal_formula*
| *dia* : *modal_formula* → *modal_formula*.

Here the syntax of a modal formula is defined. For example, *or* (*atom* *a*) (*atom* (*S* *a*)) is a syntactically correct modal formula. As is *implies* (*dia* (*atom* (*S* *a*))) (*atom* *a*).

5.3.3 Notation

Infix "∧" := and (at level 79).
Infix "∨" := or (at level 79).
Infix "⇒" := implies (at level 99).
Notation "⊥" := falsum.
Notation "¬ *A*" := (*A* ⇒ ⊥) (at level 74).
Notation "◊" := dia.
Notation "□" := box.

Here notation is introduced so that formulas can be written more naturally. Now we can write (*atom* *a*) ∨ (*atom* (*S* *a*)) and also (◊ (*atom* (*S* *a*))) ⇒ (*atom* *a*).

5.3.4 Using Coq's logic as a meta-language

In Coq, logical statements have the type `Prop`. So in order to use Coq's logic as a meta-language, a predicate of type *modal_formula* → `Prop` is needed. This is declared as a `Parameter` (or `Axiom`) which essentially declares that there is a predicate in the Coq meta-language that describes which *modal_formula*s are true. At this stage no *modal_formula*s are considered true, it is only when the axioms below are added that certain *modal_formula*s are considered true.

`Parameter` *True* : *modal_formula* → `Prop`.

5.3.5 Axioms in Coq

Here the same axioms as earlier are defined, however these axioms are now written in Coq. There are other axioms that would be necessary for fully describing CS4 and intuitionistic logic but these have been omitted for brevity and only the axioms relevant to the proof below have been included.
These axioms declare what is provable in the Coq meta-language with respect to the predicate *True*.

`Axiom` *A1* : ∀ *A B C*, *True* (((*A* ⇒ *B*) ⇒ ((*B* ⇒ *C*) ⇒ (*A* ⇒ *C*)))).
`Axiom` *diaT* : ∀ *A*, *True* (*A* ⇒ (◊ *A*)).
`Axiom` *property11_p24* : ∀ *A B*, *True* (((*A* ⇒ *B*) ⇒ (¬ *B* ⇒ ¬ *A*))).
`Axiom` *mp* : ∀ {*A*} {*B*}, *True* (*A* ⇒ *B*) → (*True* *A*) → *True* *B*.

5.4 Proof of Martin-Löf's Third Law

Martin-Löf's Third Law states that *"From the unknowability of the truth of a proposition, its falsity may be inferred"* [5, p. 12]. Using $\Diamond$ to represent 'can be known', we can formulate the Third Law as $((\neg (\Diamond p))) \Rightarrow ((\Diamond (\neg p)))$. Below is a Coq proof of the Third Law formulated in a way that mirrors the structure of Marcus' proof from earlier in this chapter.

5.4.1 Proof mirroring the structure of Marcus' proof

```
Lemma mlThirdLaw_forwards_reasoning : ∀ p, True (((¬ ( ◇ p))) ⇒ (( ◇ (¬ p)))).
Proof.
intro p.
pose proof (A1 (¬ ( ◇ p)) (¬ p) ( ◇ (¬ p)) ) as Line1.
pose proof (diaT p) as Line2.
pose proof (property11_p24 p ( ◇ p)) as Line3.
pose proof (mp Line3 Line2) as Line4.
pose proof (mp Line1 Line4) as Line5.
pose proof (diaT ( ¬ p)) as Line6.
pose proof (mp Line5 Line6) as Line7.
exact Line7.
Qed.
```

At each line in the above proof, the output from Coq when stepped through interactively is exactly each line from Marcus' original proof. For instance after "`pose proof (diaT p) as Line2.`" the output is "*Line2* : *True* $(p \Rightarrow \Diamond p)$". The reasoning is also mirrored, with *Line2* in Coq containing "*diaT*" where Marcus' proof contained "$\Diamond$ T CS4 axiom" for example.

5.4.2 Alternate proof of Martin-Löf's Third Law

Here an alternate version of the above proof is given. It uses a more typical Coq proof structure, reasoning backwards from the goal rather than takings steps towards the goal.

```
Lemma mlThirdLaw_backwards_reasoning : ∀ p, True (((¬ ( ◇ p))) ⇒ (( ◇ ( ¬ p)))).
Proof.
intros.
apply @mp with (A:=( ¬ p ⇒ ◇ (¬ p))).
 - apply @mp with (A:=( ¬ (◇ p)) ⇒ (¬ p)) .
     + apply A1.
     + apply @mp with (A:= (p ⇒ ◇ p)).
         × apply property11_p24.
         × apply diaT.
 - apply diaT.
Qed.
```

5.5 Reasonableness of the axioms

The purpose of considering the statements and lemmas below is to test whether "can be known" is an adequate interpretation for $\Diamond$.

5.5.1 Considering the axioms themselves

`Axiom` *diaT* : $\forall$ *A*, *True* $(A \Rightarrow (\Diamond\ A))$.
This axiom claims that for all propositions in CS4 and intuitionistic logic, if they are true then they can be known. This seems reasonable, however at the same time it also seems very close to saying there are no undecidable propositions (if everything that is true can be known then surely nothing is unknowable). So perhaps the axiom is stating too much.

`Axiom` *A1* : $\forall$ *A B C*, *True* ($((A \Rightarrow B) \Rightarrow ((B \Rightarrow C) \Rightarrow (A \Rightarrow C))))$.
This axiom appears to be similar to the cut elimination rule. It seems acceptable and does not directly refer to $\Diamond$. It appears to be unlikely to cause any issues relating to the interpretation of $\Diamond$ as "can be known".

`Axiom` *property11_p24* : $\forall$ *A B*, *True* ($((A \Rightarrow B) \Rightarrow (\neg\ B \Rightarrow \neg\ A))$).
This axiom, stating that an implication implies its contrapositive is unlikely to have any impact on the reasonableness of treating $\Diamond$ as "can be known".

`Axiom` *mp* : $\forall$ $\{A\}$ $\{B\}$, *True* $(A \Rightarrow B) \rightarrow$ (*True* *A*) $\rightarrow$ *True* *B*.
The axiom of Modus Ponens is very unlikely to affect the reasonableness of treating $\Diamond$ as "can be known".

So, the main axiom that requires consideration is *diaT* which states $\forall$ *A*, *True* $(A \Rightarrow (\Diamond\ A))$.

5.5.2 Considering some lemmas

The goal of considering these lemmas is to test the reasonableness of treating $\Diamond$ as "can be known". The focus will be on the *diaT* axiom which was identified above as the axiom which is likely to require the most consideration.

`Lemma` *Lemma_one* : $\forall$ *A*, *True* $(((\Diamond\ (\Diamond\ (\Diamond\ A)))) \Rightarrow (\Diamond\ (\Diamond\ (\Diamond\ (\Diamond\ A)))))$.
`Proof. intro. apply` *diaT*. `Qed.`

This lemma states that for all propositions, if it can be known that it can be known that it can be known then it can be known that it can be known that it can be known that it can be known. More generally, it should be provable that for any length of n "can be known" repeats one further repeat would also be true. This seems reasonable, although the case *A* $\Rightarrow \Diamond\ A$ still may be stating too much. For this lemma however, treating $\Diamond$ as "can be known" seems to be fine.

`Lemma` *Lemma_two* : $\forall$ *A*, *True* ($\neg\ A \Rightarrow \Diamond\ (\neg\ A)$).
`Proof. intro. apply` *diaT*. `Qed.`

This lemma states that for all propositions, if their negation is true then the negation can be known. This seems to fit well with the interpretation of $\Diamond$ as "can be known". However it may be stating too much. $\neg\ A$ being true is essentially the definition of A being false and so this lemma states that if any statement is false then it is knowable that it is false. This seems to indicate that the axiom *diaT* is too strong.

5.6 Extending to Martin-Löf's Theorem

In order to prove Martin-Löf's Theorem, two additional axioms are required. They are similar to Fact 2 and Fact 4 from Martin-Löf's argument as discussed in chapter 2 and in the Logicomp 301 slides [6].
Fact 2 states: *"If A can be known to be true and B can be known to be true, then $A \wedge B$ can be known to be true."* It turns out that slightly altered versions of Fact 2 and Fact 4 are required. The axioms needed are more basic. Informally, for Fact 2 we instead have *"If A is true and B is true, then $A \wedge B$ is true."*.

`Axiom` *fact_2* : $\forall\ A\ B, (True\ A \wedge True\ B) \leftrightarrow True\ (A \wedge B)$.

Fact 4 states: *"One and the same proposition A cannot both be known to be true and be known to be false."*. Instead we have *"For any proposition A, it is not the case that $A \wedge (\neg\ A)$ is true."*.

`Axiom` *fact_4* : $\forall\ A, \neg\ (True\ (A \wedge (\neg\ A)))$.

Note here that the outermost $\neg$ is in the Coq meta-language and captures the meaning of "cannot" in Fact 4, whereas the innermost $\neg$ is part of a *modal_formula* and captures the meaning of "to be false".
Both Fact 2 and Fact 4 are now axioms added to Coq (as *fact_2* and *fact_4*). They are needed in the proof of Martin-Löf's Theorem, *ml_theorem_backwards_reasoning/forwards_reasoning*, below.
Fact 1 is Martin-Löf's Third Law which states: *"From the unknowability of the truth of a proposition, its falsity may be inferred"*. This has been proven earlier.

`Definition` *fact_1* := *mlThirdLaw_forwards_reasoning*.

Here is a Coq-style proof of Martin-Löf's Theorem which states *"There is no proposition which can neither be known to be true nor be known to be false, i.e. there is no absolutely unprovable proposition."*

```
Theorem ml_theorem_backwards_reasoning : ∀ A, ¬ True (( ¬ (◇ A) ∧ (¬ (◇ (¬ A))))).
Proof.
intro.
unfold not.
intro.
apply fact_2 in H. destruct H.
apply (fact_4 (◇ (¬ A))).
apply fact_2.
```

`exact` ((*conj* (*mp* (*mlThirdLaw_forwards_reasoning* *A*) *H*) *H0*)).
`Qed.`

Axioms used in the above proof:

property11_p24 : $\forall$ *A B* : *modal_formula*, *True* ((*A* $\Rightarrow$ *B*) $\Rightarrow$ ($\neg$ *B* $\Rightarrow$ $\neg$ *A*))
mp : $\forall$ *A B* : *modal_formula*, *True* (*A* $\Rightarrow$ *B*) $\rightarrow$ *True A* $\rightarrow$ *True B*
fact_4 : $\forall$ *A* : *modal_formula*, $\neg$ *True* (*A* $\wedge$ $\neg$ *A*)
fact_2 : $\forall$ *A B* : *modal_formula*, *True A* $\wedge$ *True B* $\leftrightarrow$ *True* (*A* $\wedge$ *B*)
diaT : $\forall$ *A* : *modal_formula*, *True* (*A* $\Rightarrow$ $\Diamond$ *A*)
True : *modal_formula* $\rightarrow$ `Prop`
A1 : $\forall$ *A B C* : *modal_formula*, *True* ((*A* $\Rightarrow$ *B*) $\Rightarrow$ ((*B* $\Rightarrow$ *C*) $\Rightarrow$ (*A* $\Rightarrow$ *C*)))

Here is another proof of the Martin-Löf Theorem which has a more similar structure to the informal proof given in the Logicomp 301 slides [6].

`Lemma` *ml_theorem_forwards_reasoning* : $\forall$ *A*, $\neg$ *True* (($\neg$ ($\Diamond$ *A*) $\wedge$ ($\neg$ ($\Diamond$ ($\neg$ *A*))))).
`Proof.`
`intro.`

"If *A* is a proposition which cannot be known to be true, then by Fact 1, *A* can be known to be false, ..."

`assert` (*True* ($\neg$ ($\Diamond$ *A*) $\Rightarrow$ ($\Diamond$ ($\neg$ *A*)))). `apply` *fact_1.*
`unfold` *not.* `intro.` `apply` *fact_2* `in` *H0.* `destruct` *H0.* `apply` (*mp H*) `in` *H0.*

"... a contradiction."

`pose` *proof* (*fact_2* ($\Diamond$ ($\neg$ *A*)) ($\neg$ ($\Diamond$ ($\neg$ *A*)))).
`pose` *proof* (*conj H0 H1*). `apply` *H2* `in` *H3.*
`apply` *fact_4* `in` *H3.*
contradiction.
`Qed.`

The two proofs above prove the Martin-Löf theorem in this system, CS4 + intuitionistic propositional logic + *fact_2* + *fact_4*. It should be possible to derive *fact_2* and *fact_4* from intuitionistic logic, this could be verified in the future.

Chapter 6

Formalising the Martin-Löf Theorem - Formalisation of Intuitionistic Logic

6.1 Motivation and background

The Martin-Löf theorem involves intuitionistic logic and so a technique for formalising the theorem could be to formalise an embedding of intuitionistic logic in Coq and then use Coq as a meta-language to prove results about intuitionistic logic. This technique is explored in this chapter. The Third Law and the Martin-Löf Theorem are both proved in this formalisation. This technique has some similarities to the bi-modal logic embedding in Coq in that an embedding of a logic is defined with Coq as the meta-language.

6.2 Defining an embedding of intuitionistic logic

6.2.1 Atoms

Here we define a countably infinite set of atoms.

`Inductive` *Atoms* :=
| a : *Atoms*
| S : *Atoms* → *Atoms*.

`Parameter` *AtomValuation* : *Atoms* → *bool*. Here we assume a valuation for the atoms exists.

6.2.2 Propositions

Here we define the syntax of a *proposition*.

`Inductive` *proposition* :=
| *Atom* : *Atoms* → *proposition*
| *Implies* : *proposition* → *proposition* → *proposition*

| *Or* : *proposition* → *proposition* → *proposition*
| *And* : *proposition* → *proposition* → *proposition*
| ⊥ : *proposition*
| *Not* : *proposition* → *proposition*
| *K* : *proposition* → *proposition*. "*K*" for representing 'can be known' or 'knowable' is included here.
No notations have yet been defined so each *proposition* must be written in prefix notation, for example *Implies* (*S S a*) (*And a* (*S a*)) rather than (*S S a*) → (*a* ∧ (*S a*)).

6.2.3 Proofs

Next we define what constitutes a proof of a *proposition*.
Due to complications discussed below, the rules for what constitutes a proof of an implication must be defined in a roundabout manner. First, we assume the existence of a predicate in the Coq meta-language of arity 2 named *Implication_Is_True*. An axiom will be added later to define the meaning of *Implication_Is_True*.

`Parameter` *Implication_Is_True* : *proposition* → *proposition* → `Prop`.

Now we define what constitutes a proof of a *proposition*. By using an inductively defined proposition, the notion that these rules are the only possible ways to obtain a proof is captured.

`Inductive Proof` : *proposition* → `Prop` :=
| *atom_ev* : ∀ (*A*:*Atoms*), *AtomValuation A* = *true* → `Proof` (*Atom A*)
| *and_ev* : ∀ (*p q* : *proposition*), `Proof` *p* → `Proof` *q* → (`Proof` (*And p q*))
| *orl_ev* : ∀ (*p q* : *proposition*), `Proof` *p* → (`Proof` (*Or p q*))
| *orr_ev* : ∀ (*p q* : *proposition*), `Proof` *q* → (`Proof` (*Or p q*))
| *not_ev* : ∀ (*p* : *proposition*), `Proof` (*Implies p* ⊥) → (`Proof` (*Not p*))
| *K_ev* : ∀ (*p* : *proposition*), `Proof` *p* → (`Proof` (*K p*))
| *implies_ev* : ∀ (*p q* : *proposition*), *Implication_Is_True p q* → (`Proof` (*Implies p q*)).

`Axiom` *implies_ev'* : ∀ *p q*, *Implication_Is_True p q* ↔ (∃ (*f*: `Proof` *p* → `Proof` *q*), *True*).
Here we declare that a *proposition* of the form *Implies a b* has a proof iff there exists a function which takes a proof of *a* to a proof of *b*. This defines the meaning of *Implication_Is_True*.
The rule for *Implies* is added as an extra axiom because Coq gives the error "Non strictly positive occurrence ..." when trying to add it to the above definition of `Proof`. Further work is needed to ensure that adding this axiom (*implies_ev'*) and the *Implication_Is_True* predicate as done here does not give rise to a contradiction or other unintended consequences.

In a similar way to the bi-modal logic embedding, the definitions required to embed intuitionistic logic are now all defined. Only notation, lemmas and theorems are defined below and these do not alter what propositions are expressible and provable in this embedding of intuitionistic logic in Coq.

6.2.4 Notation

Infix "$\wedge$" := And (at level 79).
Infix "$\vee$" := Or (at level 79).
Infix "$\Rightarrow$" := Implies (at level 99).
Notation "$\bot$" := Falsum.
Notation "$\neg$ A" := ($A \Rightarrow \bot$) (at level 74).

6.2.5 Lemmas

With an embedding of intuitionistic logic now defined, we can prove some relevant results.

`Lemma` *falsum_unprovable* : $\neg$ (`Proof` $\bot$).
`Proof`.
`unfold` *not*.
`intros`.
`inversion` *H*. In our inductive definition for `Proof` we did not include a case where $\bot$ was provable. As a result we can use the tactic `inversion` to obtain a contradiction from a hypothetical Coq proof of `Proof` $\bot$.

`Qed`.

Using similar methods we can prove that we cannot have a proof of a proposition and its negation.

`Lemma` *non_contradiction* : $\forall$ A, $\neg$ (`Proof` $A \wedge$ `Proof` (*Not* A)).
`Proof`.
`unfold` *not*.
`intros`.
`destruct` *H*.
`inversion` *H0*.
`inversion` *H2*.
`apply` *implies_ev'* `in` *H4*.
`destruct` *H4*.
`apply` *falsum_unprovable*.
`apply` *x*.

```
apply H.
Qed.
```

Here Fact 2 from Cristian Calude's explanation of the Martin-Löf Theorem in chapter 2 is proven. Fact 2 states that "If A can be known to be true and B can be known to be true, then $A \wedge B$ can be known to be true." [6]

```
Lemma fact_2 : ∀ A B, Proof (K A) ∧ Proof (K B) → Proof (K (A ∧ B)).
Proof.
intros.
destruct H.
apply K_ev.
apply and_ev.
inversion H. exact H2.
inversion H0. exact H2.
Qed.
```

It will also be useful to prove the similar statement *fact_2'* which states that for all propositions A and B, there is a proof for A and a proof for B if and only if there is a proof for $(A \wedge B)$.

```
Lemma fact_2' : ∀ A B, Proof (A) ∧ Proof (B) ↔ Proof ((A ∧ B)).
Proof.
intros.
split.
 - intro.
   destruct H.
   apply and_ev.
     exact H.
     exact H0.
 - intro.
   inversion H.
   split.
   exact H2.
   exact H3.
Qed.
```

Here is a proof of Fact 3, which states that "The proposition $\perp$ cannot to be known to be true." [6]

```
Lemma fact_3 : ¬ Proof (K ⊥).
Proof.
unfold not. intros.
inversion H.
inversion H1.
Qed.
```

Here Fact 4 is proven. It states "One and the same proposition A cannot both be known to

be true and be known to be false." [6]

```
Lemma fact_4 : ∀ A, ¬ ((Proof (K A)) ∧ Proof (K ( ¬ A))).
Proof.
intro.
unfold not.
intro.
destruct H.
inversion H. clear H1. clear p.
inversion H0. clear H1. clear p.
apply not_ev in H3.
apply (non_contradiction A).
split. exact H2. exact H3.
Qed.
```

6.3 Proof of Martin-Löf's Third Law

```
Lemma ml3rdLaw : ∀ p, Proof (Not (K p)) → Proof (K (Implies p ⊥)) .
Proof.
intro p.
intros.
apply K_ev. apply implies_ev. apply implies_ev'.
inversion H. inversion H1. apply implies_ev' in H3.
destruct H3.
assert(Proof p → Proof (K p)).
intros. apply K_ev in H5. exact H5.
assert(Proof p → Proof ⊥).
intros.
apply x.
apply H5.
exact H6.
∃ H6.
apply I.
Qed.
```

6.4 Proof of the Martin-Löf Theorem

We can now prove the Martin-Löf Theorem which states, in one rendering, that "it is impossible to find a counterexample to the law of the excluded middle in its positive formulation" [5, p. 14]. Here this is rendered as 'for all propositions, it is not the case that it is provable that the proposition both cannot be known to be true and cannot be known to be false'.

```
Theorem MLTheorem : ∀ A, ¬ Proof (( ¬ (K A) ∧ ( ¬ (K ( ¬ A))))).
```

```
Proof.
intro.
unfold not.
intro.
apply fact_2' in H. destruct H.
apply (non_contradiction (K ( ¬ A))).
split.
 - apply ml3rdLaw.
   apply not_ev.
   exact H.
 - apply not_ev.
   exact H0.
Qed.
```

6.5 Comments on this formalisation

A key advantage of this formalisation is that it does not rely on axioms other than those fundamental to intuitionistic logic. The only two non-standard axioms are found in the definition of what constitutes a proof in the system.
K_ev is essentially an axiom which states that

$$\forall(p : proposition), \texttt{Proof}\, p \rightarrow (\texttt{Proof}(Kp)).$$

This potentially does not capture the notion of knowability properly. It is part of an inductive definition, so implicit in the axiom is that the *only* way one can show that a proposition is knowable is by exhibiting a Coq proof for `Proof` p. It may be that there are other ways of knowing that are not captured by this axiom. For instance we may be able to know by analysing the axioms of intuitionistic logic that double negation elimination, $\neg \neg q \rightarrow q$, is not knowable, yet we would not be able to prove `Proof` $\neg (K (\neg (\neg q) \rightarrow q))$.
The second non-standard axiom is that the only way to show that `Proof` $(A \Rightarrow B)$ is true is to show that $(\exists$ (f: `Proof` $p \rightarrow$ `Proof` q), *True*). This can be simplified to `Proof` $p \rightarrow$ `Proof` q. A possible concern with this axiom is it allows `Proof` $(A \Rightarrow B)$ to be proven whenever the implication `Proof` $p \rightarrow$ `Proof` q is true rather than when there is indeed a function that takes a proof of p and returns a proof of q. This means that alterations to the meta-language logic could sometimes filter down into the embedding of intuitionistic logic, potentially causing issues.
Overall however, this formalisation does a good job at capturing certain elements from Martin-Löf's proof and is useful for gaining a deeper understanding of the Martin-Löf Theorem.

Chapter 7

Open Questions

7.1 Double negation elimination and the Martin-Löf Theorem

In chapter 4, double negation elimination was considered with regards to the completeness theorems. ($\neg\neg q \to q$) also has interesting considerations in relation to the Martin-Löf Theorem.
For instance, Martin-Löf claims that "it is impossible to find a counterexample to the law of the excluded middle in its positive formulation" [5, p. 14]. If we consider

$$(\neg\neg q \to q) \vee (\neg(\neg\neg q \to q))$$

we would think this must not be true, since neither disjunct is provable intuitionistically. However this would then count as "a counterexample to the law of the excluded middle in its positive formulation".
What then resolves this? A possible answer is that the reasoning used to justify that neither disjunct is provable intuitionistically is not itself intuitionistic reasoning. That is, analysing the axioms of intuitionistic logic to conclude that ($\neg\neg q \to q$) is not intuitionistically provable does not give us a canonical proof of $\bot$ from ($\neg\neg q \to q$). This means that although we *know* neither disjunct is provable we do not know this intuitionistically and thus we do not have "a counterexample to the law of the excluded middle in its positive formulation" after all.
Nevertheless, this example does show that care needs to be taken when interpreting the implications of the Martin-Löf Theorem.

7.2 Simplifying the axioms in the CS4 + Int formalisation

In the proof of Martin-Löf's Theorem in chapter 5, both Fact 2 and Fact 4 are added as axioms. It may be possible however to derive these facts from more fundamental rules for

intuitionistic logic, in a similar way to how this is done in the intuitionistic logic formalisation in chapter 6.
The advantage of this would be that the trustworthiness of the proof of the Martin-Löf Theorem would rest upon the fundamental rules rather than the correctness of Fact 2 and Fact 4. However both Fact 2 and Fact 4 (shown below) are fairly primitive axioms so this is not too great a concern.

`Axiom` *fact_2* : $\forall\ A\ B, (\mathit{True}\ A \wedge \mathit{True}\ B) \leftrightarrow \mathit{True}\ (A \wedge B)$.
`Axiom` *fact_4* : $\forall\ A, \neg\ (\mathit{True}\ (A \wedge (\neg\ A)))$.

7.3 Implications of the 'Non-strictly positive occurrence' error in the intuitionistic logic formalisation

In the definition of what constitutes a proof in the intuitionistic logic embedding in chapter 6 the most natural definition for the proof rule for implication results in the Coq error 'Non strictly positive occurrence of "Proof" in "$\forall\ p\ q$: *proposition*, (`Proof` $p \rightarrow$ `Proof` q) $\rightarrow$ `Proof` (*Implies* $p\ q$)" '. A shorter example of a definition that gives the same error is shown below. From further reading [22, 23] it does unfortunately seem that that circumventing the error as done in the intuitionistic logic formalisation is very likely to enable $\bot$ to be provable. Further work would be needed to ensure this is the case and to check whether a slightly altered definition for *implies_ev* might be able to avoid the issue. Also, the proofs in the chapter (as far as I am aware) do not make use of the flaw that is introduced and so it is possible that the reasoning in the chapter is still useful.

`Inductive Proof :` *proposition* $\rightarrow$ `Prop` :=
  | *implies_ev* : $\forall\ (p\ q$: *proposition*), (`Proof` $p \rightarrow$ `Proof` $q) \rightarrow$ (`Proof` (*Implies* $p\ q$)).

Bibliography

[1] P. Martin-Löf, "Verificationism then and now," in *The Foundational Debate*, pp. 187–196, Springer, 1995.

[2] N. Alechina, M. Mendler, V. De Paiva, and E. Ritter, "Categorical and Kripke semantics for constructive S4 modal logic," in *International Workshop on Computer Science Logic*, pp. 292–307, Springer, 2001.

[3] "The Coq proof assistant." https://coq.inria.fr/

[4] H. Herbelin, F. Kirchner, B. Monate, and J. Narboux, "Coq version 8.0 for the clueless." http://flint.cs.yale.edu/cs428/coq/doc/faq.html#htoc7

[5] P. Martin-Löf, "Verificationism then and now," in *Judgement and the Epistemic Foundation of Logic*, pp. 3–14, Springer, 2013.

[6] C. S. Calude, "Logicomp 301 slides." http://www.cs.auckland.ac.nz/~cristian/301/301_beamer.pdf

[7] H. Wang, *From mathematics to philosophy*. International library of philosophy and scientific method, London, New York: Routledge & Kegan Paul; Humanities Press, 1974.

[8] K. Godel, "Collected works, vol. II," *New York: Oxford UP*, 1990.

[9] K. Gödel, "Some basic theorems on the foundations of mathematics and their implications," *Collected Words*, pp. 304–323, 1995.

[10] A. Urquhart, "Emil post.," *Logic from Russell to Church*, vol. 5, pp. 617–666, 2009.

[11] E. L. Post, "Formal reductions of the general combinatorial decision problem," *American journal of mathematics*, vol. 65, no. 2, pp. 197–215, 1943.

[12] S. Feferman, "Are there absolutely unsolvable problems? Gödel's dichotomy," *Philosophia Mathematica*, vol. 14, no. 2, pp. 134–152, 2006.

[13] C. Benzmüller and B. W. Paleo, "Interacting with modal logics in the Coq proof assistant," in *International Computer Science Symposium in Russia*, pp. 398–411, Springer, 2015.

[14] M. Fitting and R. L. Mendelsohn, *First-order modal logic*, vol. 277. Springer Science & Business Media, 2012.

[15] D. Thompson, "Natural deduction planner." https://sourceforge.net/projects/proofassistant/

[16] J. Moschovakis, "Intuitionistic logic," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, spring 2015 ed., 2015.

[17] "Gödel's completeness theorem - Wikipedia." https://goo.gl/edJgsW

[18] M. L. Schagrin and H. Wang, "Metalogic." https://goo.gl/nfaLux, Jan 2011.

[19] D. Van Dalen, *Logic and structure*. Springer, 2004.

[20] S. Shapiro, "Classical logic," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, winter 2017 ed., 2017.

[21] S. B. University, "Introduction to intuitionistic logic." http://www3.cs.stonybrook.edu/~pfodor/courses/CSE371/slides10/10slides.pdf

[22] V. Sjöberg, "Why must inductive types be strictly positive?." http://vilhelms.github.io/posts/why-must-inductive-types-be-strictly-positive/

[23] J. Gross, "Would it be inconsistent to relax Coq's strict positivity checker to not look at type indices of the inductive type being defined?." https://goo.gl/Bt3UwL, Jan 2018.