Formalising the Completeness Theorem of Classical Propositional Logic in Agda (Proof Pearl)

Leran Cai, Ambrus Kaposi, and Thorsten Altenkirch

University of Nottingham
{psylc5, psxak8, psztxa}@nottingham.ac.uk

Abstract. A computer formalisation of the completeness of the boolean model of classical propositional logic is presented. The work follows Huth and Ryan's proof [9]. The proof is constructed for a classical logic system in natural deduction style with all logical connectives. The formalisation is constructive and uses the interactive theorem prover Agda which is an implementation of intensional Martin-Löf type theory [11]. Functions have to be defined in a structurally recursive way to pass the termination checker of Agda. The basic definitions of the formal system must be carefully chosen in order to provide a convenient environment to build correct proofs and meanwhile prevent from getting warnings from the type checker. The formalisation is written in an accessible way so that it can be used for educational purposes. The full source code is available online[1].

Keywords: Agda, Completeness Theorem, Classical Propositional Logic

1 Introduction

1.1 Outline

This paper describes the implementation of the completeness theorem for classical propositional logic in Agda. By interactively implementing the proof in Agda, we can attain deeper understanding of how to construct formal proofs for arbitrary propositions from their propositional atoms and how the law of the excluded middle (LEM) is used. Informally, the completeness theorem can be stated as follows:

(Soundness) If a propositional formula has a proof deduced from the given premises, then all assignments of the premises which make them evaluate to true also make the formula evaluate to true.

[1] The formalisation is available at `http://bitbucket.org/Leran/propositional-logic`. It has been typechecked with Agda version 2.4.2.2 and standard library 0.9.

(Completeness) If for all assignments of the axioms and premises which make them evaluate to true also make a propositional formula true, then it is always possible to construct a proof of this formula by applying the deduction rules on the given premises.

The name of the second theorem alone is completeness theorem which is confusing because the general completeness theorem usually claims them both [6] [10]. Thus we call the theorems as the soundness part and completeness part of the completeness theorem.

1.2 Related Work

The context of this work is formalising completeness theorems of formal logic systems in Agda. Formalising the completeness of intuitionistic logic with Kripke model in a proof assistant has been finished by a number of researchers. Herbelin and Lee presented an implementation of the completeness of Kripke model for intuitionistic logic with implication and universal quantification in Coq [8]. Blanchette et al. construct the completeness theorem of first-order logic by using Gentzen's system and codatatypes [4]. Coquand presents a fully formalised proof of the soundness and completeness of simply-typed λ-calculus with respect to Kripke models [5].

We focus on classical propositional logic in natural deduction style and prove its soundness and completeness with regards the boolean model. In the future, we will attempt to formalise other completeness theorems for nonstandard Kripke model and Beth model of the intuitionistic full first-order predicate logic ([15], [13]).

The formalisation takes place in a constructive metalanguage of Agda, which means that the proof has computational content. The corresondance between classical logic and computation has been first noticed by Griffin [7]. A good summary of the developments in this area is Urban's PhD thesis [14].

2 Background

2.1 Agda

All proofs were conducted in Agda ([11], [1]), an interactive proof assistant which implements Martin-Löf type theory [12] and can be used as a framework to formalise formal logic systems. It is also a dependently typed functional programming language with inductive families. Π types (dependent functions) are written by the generalised arrow notation $(\mathsf{x{:}A}) \to \mathsf{B}$ which can be interpreted as $\forall x \in A.B$ by the Curry-Howard correspondance. Implicit arguments are denoted by curly brackets eg. $\mathsf{f} : \{\mathsf{x{:}A}\} \to \mathsf{B}$, in this case it is not necessary to provide these arguments as Agda will try to deduce them. We can still provide them by the $\mathsf{f}\ \{\mathsf{x} = \ldots\}$ notation. The notation $\forall\ \{\mathsf{x}\}$ abbreviates $\{\mathsf{x} : \_\}$ where the type of x is inferred by Agda, the underline character denoted a wildcard. Agda functions can be defined with infix or mixfix notation, eg. $\_\Rightarrow\_$ is

an infix function, ~_ is a prefix operator. The underlines denote the positions of the arguments.

Mathematical proofs in Agda are written as structurally recursive functions. Therefore, constructing a mathematical proof in Agda is equivalent to constructing a well-typed function which can terminate. Induction is performed by pattern matching on the arguments of the function. Inductive types, and more generally, inductive families can be defined by using the **data** keyword.

In addition, Agda supports Unicode characters, which allows us to write proofs that look like common logic proofs on textbooks.

2.2 Syntax and Semantics

The formalisation of the completeness theorem requires a proper design of the classical propositional logic system in which the syntax and semantics should be carefully defined. For the syntax, a crucial question is how to define propositional formulas. A propositional formula is a string of indivisible propositional atoms and logical connectives. In Agda, we can define them as inductive types and thereby finding the main connective and making the tree structure of a formula explicit is trivial.

As for semantics, every proposition in classical propositional logic can be assigned with a boolean value. This can be implemented as a function which takes a propositional formula as its input and its output is the meaning of the formula.

With appropriate definitions, the soundness proof and completeness proof can be implemented more easily. The proofs essentially follow Huth and Ryan [9] but completes the unfinished proofs which they do not explain, especially the use of the law of the excluded middle.

2.3 Completeness theorem

In this part we summarize what the completeness theorem states and briefly introduce the ideas of their proofs. Before we formally give the completeness theorem, we need to specify how to interpret some fundamental concepts of the completeness theorem.

Definition 1. *(Sequent) A sequent is a set of formulas $\phi_1, \phi_2, ..., \phi_n$ called premises and another formula ψ which is called conclusion.*

Definition 2. *(Proof tree) A proof tree of a sequent $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ is a tree with root ψ where the nodes are propositional logic deduction rules and $\phi_1, ..., \phi_n$ can be leaves.*

Definition 3. *(Context) A context is a set of all the premises in a sequent. The sequent $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ can be written as $\Gamma \vdash \psi$ where Γ contains ϕ_i for all $1 \leqslant i \leqslant n$. When the sequent only contains a theorem as its conclusion, the context is an empty collection $\varnothing$, e.g. $\varnothing \vdash \psi$.*

Definition 4. *(Semantic entailment) If for all valuations (mappings from the propositional atoms to booleans) in which the context Γ evaluates to true, ψ also evaluates to true, we say that Γ semantically entails ψ. It can be written as $\Gamma \vDash \psi$.*

With the interpretations above, the we can state soundness and completeness:

Theorem 1. *(Soundness) Let $\phi_1, \phi_2, ..., \phi_n$ and ψ be propositional logic formulas. If $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ is valid, then $\phi_1, \phi_2, ..., \phi_n \vDash \psi$ holds.*

Theorem 2. *(Completeness) Let $\phi_1, \phi_2, ..., \phi_n$ and ψ be propositional logic formulas. If $\phi_1, \phi_2, ..., \phi_n \vDash \psi$ holds, then $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ is valid.*

The completeness theorem consists of both. Therefore, the corollary can be stated as follows:

Corollary 1. *(Completeness Theorem) Let $\phi_1, \phi_2, ..., \phi_n, \psi$ be formulas of propositional logic. $\phi_1, \phi_2, ..., \phi_n \vDash \psi$ holds iff the sequent $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ is valid.*

2.3.1 Soundness proof The soundness part asserts: $\Gamma \vdash \psi \to \Gamma \vDash \psi$. To prove it, we do induction on the depth of the proof tree of $\Gamma \vdash \psi$. In essence, this induction checks the *last propositional deduction rule* the proof of $\Gamma \vdash \psi$ uses. For example, if the least significant connective in ψ is $\wedge$ which means ψ has form $\phi_1 \wedge \phi_2$, then we can say that the last rule in the proof of $\Gamma \vdash \psi$ is conjuction ($\wedge$) introduction rule which is the only way to construct propositions which have the form $\phi_1 \wedge \phi_2$. Then we use the induction hypothesis to prove that $\Gamma \vDash \phi_1$ and $\Gamma \vDash \phi_2$ both hold and use them to prove $\Gamma \vDash \phi_1 \wedge \phi_2$ ($\Gamma \vDash \psi$) by discussing how logic connectives handle truth values. In Agda, the induction on the depth of the proof tree becomes structural induction defined by pattern matching.

2.3.2 Completeness proof The completeness part asserts: $\Gamma \vDash \psi \to \Gamma \vdash \psi$ where Γ consists of $\phi_1, \phi_2, ..., \phi_n$. This theorem should be proved in three steps:

Lemma 1. Our first step is to prove the theorem: $\phi_1, \phi_2, ..., \phi_n \vDash \psi \to \varnothing \vDash \phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi$. This states that if $\phi_1, \phi_2, ..., \phi_n$ semantically entails ψ then $\phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi$ is a *tautology*.

Lemma 2. In this step, the *weak version* of completeness which is $\varnothing \vDash \eta \to \varnothing \vdash \eta$ is proved. This asserts that *all tautologies have a proof*. By proving this, we can demonstrate that $\varnothing \vDash \phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi \to \varnothing \vdash \phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi$.

Lemma 3. The last step is: $\varnothing \vdash \phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi \to \phi_1, \phi_2, ..., \phi_n \vdash \psi$. Given $\varnothing \vdash \phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi$, we add in $\phi_1, \phi_2, ..., \phi_n$ as premises and we use implication ($\Rightarrow$) elimination rule to reduce the proposition from $\phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi$ to ψ. In the end we can obtain $\phi_1, \phi_2, ..., \phi_n \vdash \psi$.

By combining these three proofs together we can prove the completeness.

3 Implementation in Agda

In this section, we first follow the definitions in the previous section to implement the syntax and semantics for our formal system. Then the detailed discussion of the proof of completeness theorem will be given as well as the code implementation in Agda.

We use the following notational conventions:

Γ	contexts
ϕ_i, ψ, χ, η	propositions
p_i	propositional atoms
ρ	valuation
σ, σ_i, δ	proof trees

3.1 Syntax

The basic object in classical propositional logic is the well-formed propositional formula. We define the inductive type Props whose inhabitants are well-formed propositions. Using the constructors $\bot$, $\top$, patom, ... is the only way to construct an object which has type Props.

```
data Props : Set where
  ⊥ ⊤          : Props
  patom        : Fin n → Props
  ~_           : Props → Props
  _∨_ _∧_ _⇒_ : Props → Props → Props
```

The number of propositional atoms is denoted n, this is a natural number parameter of the whole formalisation. Thus, a formula can refer to n different propositional atoms, this is expressed by the patom constructor which takes a value of type Fin n. For a natural number n the datatype Fin n is the type of natural numbers less than n.

For example, the formula $p_0 \Rightarrow (p_0 \vee \sim p_1)$ which mentions on 2 propositional atoms is represented by patom zero ⇒ (patom zero ∨ ∼ patom (suc zero)). patom zero is the first propositional atom.

A *context* is defined as a list where one can insert new elements at the end. In Agda, we call it Cxt. It has one natural number index, this carries the length of the context.

```
data Cxt : ℕ → Set where
  ∅   : Cxt zero
  _•_ : {l : ℕ} → Cxt l → Props → Cxt (suc l)
```

Then we can use Cxt and Props to define the type of proof trees. An element of the data type $\Gamma \vdash \phi$ will be the a proof tree which has ϕ at its root and the assumptions in Γ at its leaves. The constructors represent the deduction rules.

In this way, we can guarantee that every proof tree in Agda is valid since they can only be constructed by using deduction rules.

We write the types of the constructors in a deduction rule style, the horizontal lines are just comments in Agda.

```
data _⊢_ : {l : ℕ} (Γ : Cxt l) (ψ : Props) → Set where
  var    : ∀ {l} {Γ : Cxt l} {ψ}       → Γ • ψ ⊢ ψ
  weaken : ∀ {l} {Γ : Cxt l} {ϕ ψ}     → Γ ⊢ ψ
                                        -- ———————
                                        → Γ • ϕ ⊢ ψ
  ⊤-i    : ∀ {l} {Γ : Cxt l}           → Γ ⊢ ⊤
  ⊥-e    : ∀ {l} {Γ : Cxt l} {ψ}       → Γ ⊢ ⊥
                                        -- ———
                                        → Γ ⊢ ψ
  ~-i    : ∀ {l} {Γ : Cxt l} {ψ}       → Γ • ψ ⊢ ⊥
                                        -- ———————
                                        → Γ ⊢ ~ ψ
  ~-e    : ∀ {l} {Γ : Cxt l} {ψ}       → Γ ⊢ ψ → Γ ⊢ ~ ψ
                                        -- ———————————
                                        → Γ ⊢ ⊥
  ⇒-i    : ∀ {l} {Γ : Cxt l} {ϕ ψ}     → Γ • ϕ ⊢ ψ
                                        -- ———————
                                        → Γ ⊢ ϕ ⇒ ψ
  ⇒-e    : ∀ {l} {Γ : Cxt l} {ϕ ψ}     → Γ ⊢ ϕ ⇒ ψ → Γ ⊢ ϕ
                                        -- ———————————
                                        → Γ ⊢ ψ
  ∧-i    : ∀ {l} {Γ : Cxt l} {ϕ ψ}     → Γ ⊢ ϕ → Γ ⊢ ψ
                                        -- ———————————
                                        → Γ ⊢ ϕ ∧ ψ
  ∧-e₁   : ∀ {l} {Γ : Cxt l} {ϕ ψ}     → Γ ⊢ ϕ ∧ ψ
                                        -- ———————
                                        → Γ ⊢ ϕ
  ∧-e₂   : ∀ {l} {Γ : Cxt l} {ϕ ψ}     → Γ ⊢ ϕ ∧ ψ
                                        -- ———————
                                        → Γ ⊢ ψ
  ∨-i₁   : ∀ {l} {Γ : Cxt l} {ϕ ψ}     → Γ ⊢ ϕ
                                        -- ———————
                                        → Γ ⊢ ϕ ∨ ψ
  ∨-i₂   : ∀ {l} {Γ : Cxt l} {ϕ ψ}     → Γ ⊢ ψ
                                        -- ———————
                                        → Γ ⊢ ϕ ∨ ψ
  ∨-e    : ∀ {l} {Γ : Cxt l} {ϕ ψ χ}   → Γ ⊢ ϕ ∨ ψ
                                        → Γ • ϕ ⊢ χ
```

$$\begin{array}{lll} & \to\ \Gamma \bullet \psi \vdash \chi & \\ & \text{--}\ \overline{\qquad\qquad} & \\ & \to\ \Gamma \vdash \chi & \\ \mathsf{lem}\ :\ \forall\ \{\mathsf{l}\}\ \{\Gamma\ :\ \mathsf{Cxt}\ \mathsf{l}\}\ \{\psi\} & \to\ \Gamma \vdash \psi \vee \sim \psi & \end{array}$$

We handle variable binding by De Bruijn indices implemented by the variable rule var and the weakening rule weaken. The variable rule says that if the context ends with ψ, then we can prove ψ. The weakening rule asserts that if ψ can be deduced from the one context, then we can also deduce it from an extended context.

An example of a derivation tree is

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing \cdot \phi \wedge \psi \vdash \phi \wedge \psi}\,\text{var}}{\varnothing \cdot \phi \wedge \psi \vdash \psi}\,\wedge\text{-e}_2 \qquad \dfrac{\dfrac{}{\varnothing \cdot \phi \wedge \psi \vdash \phi \wedge \psi}\,\text{var}}{\varnothing \cdot \phi \wedge \psi \vdash \phi}\,\wedge\text{-e}_1}{\varnothing \cdot \phi \wedge \psi \vdash \psi \wedge \phi}\,\wedge\text{-i}}{\varnothing \vdash \phi \wedge \psi \Rightarrow \psi \wedge \phi}\,\Rightarrow\text{-i}$$

it is represented by the Agda term

$$\Rightarrow\text{-i}\ (\wedge\text{-i}\ (\wedge\text{-e}_2\ \mathsf{var})\ (\wedge\text{-e}_1\ \mathsf{var}))\ .$$

3.2 Semantics

In Agda, we use *semantic brackets* $[\![\_]\!]$ to give propositional formulas meanings. It is a function which takes a formula as an input and returns a boolean value as its semantics.

Any proposition either is a propositional atom or consists of a certain number of propositions and propositional connectives. Thus the semantics for an arbitrary proposition should depend on the valuation of each propositional atom and the behaviours of each propositional connective. Valuations (elements of the type Val) are functions from the n-element set to booleans.

$$\begin{array}{l} \mathsf{Val}\ =\ \mathsf{Fin}\ \mathsf{n}\ \to\ \mathsf{Bool} \\ [\![\_]\!]\ :\ \mathsf{Props}\ \to\ \mathsf{Val}\ \to\ \mathsf{Bool} \\ [\![\ \bot\ \ \ \ \ \ \ ]\!]\ \rho\ =\ \mathsf{false} \\ [\![\ \top\ \ \ \ \ \ \ ]\!]\ \rho\ =\ \mathsf{true} \\ [\![\ \mathsf{patom\ x}\ ]\!]\ \rho\ =\ \rho\ \mathsf{x} \\ [\![\ \sim \phi\ \ \ \ \ \ ]\!]\ \rho\ =\ \mathsf{neg}\ ([\![\ \phi\ ]\!]\ \rho) \\ [\![\ \phi_1 \vee \phi_2\ ]\!]\ \rho\ =\ [\![\ \phi_1\ ]\!]\ \rho\ \text{`}\mathsf{or}\text{`}\ [\![\ \phi_2\ ]\!]\ \rho \\ [\![\ \phi_1 \wedge \phi_2\ ]\!]\ \rho\ =\ [\![\ \phi_1\ ]\!]\ \rho\ \text{`}\mathsf{and}\text{`}\ [\![\ \phi_2\ ]\!]\ \rho \\ [\![\ \phi_1 \Rightarrow \phi_2\ ]\!]\ \rho\ =\ \mathsf{neg}\ ([\![\ \phi_1\ ]\!]\ \rho)\ \text{`}\mathsf{or}\text{`}\ [\![\ \phi_2\ ]\!]\ \rho \end{array}$$

In this definition, neg, and, or are the usual boolean operations.

In essence, this way of defining the semantics of propositions is the same as doing induction on syntax trees of propositions. For example, in $\phi_1 \vee \phi_2$ we

investigate the meaning of left subtree and the meaning of the right subtree. Then we use the meaning of the $\vee$ on the node to define the meaning of the whole tree. On the leaves of proposition syntax trees are all propositional atoms which are not divisible. Their semantics are determined by the valuation ρ we give.

The meaning of a context is just the conjuction of the meanings of its formulas.

```
⟦_⟧ᶜ : {l : ℕ} → Cxt l → Val → Bool
⟦ ∅ ⟧ᶜ ρ = true
⟦ Γ • ϕ ⟧ᶜ ρ = ⟦ Γ ⟧ᶜ ρ `and` ⟦ ϕ ⟧ ρ
```

Now we have defined both Props and Cxt. The semantic entailment is given as follows:

```
_⊨_ : {l : ℕ} → Cxt l → Props → Set
Γ ⊨ ψ = ∀ ρ → ⟦ Γ ⟧ᶜ ρ ≡ true → ⟦ ψ ⟧ ρ ≡ true
```

This _⊨_ relation states that for all valuations, if all formulas in the context evaluate to true then the conclusion also evaluates to true. _≡_ denotes the internal equality of Agda.

3.3 Soundness proof

We implement the soundness theorem $\Gamma \vdash \psi \to \Gamma \vDash \psi$ as the following function:

```
soundness : ∀ {l} {Γ : Cxt l} {ψ : Props} → Γ ⊢ ψ → Γ ⊨ ψ
```

If we expand the definition of $\vDash$, we get the following type:

```
soundness : ∀ {l} {Γ : Cxt l} {ψ : Props} → Γ ⊢ ψ
  → (∀ ρ → ⟦ Γ ⟧ᶜ ρ ≡ true → ⟦ ψ ⟧ ρ ≡ true)
```

As we mentioned before, we prove this by structural induction on the proof tree. By pattern matching on the proof tree σ which has type $\Gamma \vdash \psi$, we can check which is the last deduction rule. In Agda's interactive interface,

```
soundness σ = {!!}
```

becomes

```
soundness var = {!!}
soundness (weaken σ) = {!!}
soundness ⊤-i = {!!}
soundness (⊥-e σ) = {!!}
soundness (~-i σ) = {!!}
soundness (~-e σ σ₁) = {!!}
soundness (⇒-i σ) = {!!}
```

```
soundness (⇒-e σ σ₁) = {!!}
soundness (∧-i σ σ₁) = {!!}
soundness (∧-e₁ σ) = {!!}
soundness (∧-e₂ σ) = {!!}
soundness (∨-i₁ σ) = {!!}
soundness (∨-i₂ σ) = {!!}
soundness (∨-e σ σ₁ σ₂) = {!!}
soundness lem = {!!}
```

after pattern matching. The {!!} parts denote holes in the proof which need to be filled in by the user.

We exemplify filling one of the holes, the one for conjuction introduction rule. The fundamental idea is to use induction hypothesis to prove the semantic entailment relation between the context and the component of the original proposition and discuss how the logic connectives handle truth values. One important idea is that using induction hypothesis means recursively calling the *soundness* function. The decuction rule of conjunction introduction is the following:

$$\frac{\Gamma \vdash \phi_1 \qquad \Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \wedge \phi_2}\ \wedge\text{-i}$$

In the implementation, the left subtree (proving $\Gamma \vdash \phi_1$) is denoted σ_1, the right subtree (proving $\Gamma \vdash \phi_2$) is denoted σ_2.

```
soundness {ψ = ϕ₁ ∧ ϕ₂} (∧-i σ₁ σ₂) ρ ⟦Γ⟧≡true
  with ⟦ ϕ₁ ⟧ ρ | inspect ⟦ ϕ₁ ⟧ ρ | ⟦ ϕ₂ ⟧ ρ | inspect ⟦ ϕ₂ ⟧ ρ
... |    true   | [_]               | true     | [_] = refl
... |    true   | [_]               | false    | [⟦ϕ₂⟧≡false]
                  = ⟦ϕ₂⟧≡false ⁻¹ ■ (soundness σ₂ ρ ⟦Γ⟧≡true)
... |    false  | [⟦ϕ₁⟧≡false]      | _        | [_]
                  = ⟦ϕ₁⟧≡false ⁻¹ ■ (soundness σ₁ ρ ⟦Γ⟧≡true)
```

Our goal is to prove soundness when the proposition has form $\wedge$-i σ_1 σ_2 : $\Gamma \vdash \phi_1 \wedge \phi_2$. The result needs to be of type $\Gamma \vDash \phi_1 \wedge \phi_2$, that is

$$\forall\, \rho \;\rightarrow\; [\![\ \Gamma\ ]\!]^c\ \rho \;\equiv\; \mathsf{true} \;\rightarrow\; [\![\ \phi_1 \wedge \phi_2\ ]\!]\ \rho \;\equiv\; \mathsf{true}.$$

The valuation is denoted by ρ and the witness that the context evaluates to true is denoted ⟦Γ⟧=true. We prove soundness by inspecting the meaning of ϕ_1 and ϕ_2, that is pattern matching on the values of $[\![\ \phi_1\ ]\!]\ \rho$ and $[\![\ \phi_2\ ]\!]\ \rho$, respectively, by using the **with** construct of Agda. **with** is like a dependent case analysis. inspect provides witnesses of patterns matching, eg. inspect $[\![\ \phi_2\ ]\!]\ \rho$ proves $[\![\ \phi_2\ ]\!]\ \rho \equiv$ false in the case when $[\![\ \phi_2\ ]\!]\ \rho$ was matched by false.

In the case when both of meanings are true, the return type will just reduce to true ≡ true which can be proven by refl (reflexivity, the Agda constructor for the _≡_ type).

In the cases when one of them is false, we use the induction hypothesis. For example, if $[\![\ \phi_1\ ]\!]\ \rho$ is true and $[\![\ \phi_2\ ]\!]\ \rho$ is false, we need to prove false $\equiv$ true. This can be done by using transitivity (_■_) to compose the equalities

$$\phi_2 \equiv \mathsf{false}^{-1}\ :\ \mathsf{false}\ \equiv\ [\![\ \phi_2\ ]\!]\ \rho$$

and

$$\mathsf{soundness}\ \sigma_2\ \rho\ [\![\Gamma]\!] \equiv \mathsf{true}\ :\ [\![\ \phi_2\ ]\!]\ \rho\ \equiv\ \mathsf{true}.$$

The rest of the deduction rules of classical propositional logic can be proved in a similar way.

3.4 Completeness proof

In this section we present the proof of completeness and its implementation in Agda. The general theorem is $\Gamma \vDash \psi \to \Gamma \vdash \psi$. As mentioned before, the general theorem is split into three lemmas.

```
lemma1 : ∀ {I} {Γ : Cxt I} {ψ : Props} → Γ ⊨ ψ → ∅ ⊨ Γ ⇛ ψ
lemma2 : ∀ {η : Props} → ∅ ⊨ η → ∅ ⊢ η
lemma3 : ∀ {I} {Γ : Cxt I} {ψ : Props} → ∅ ⊢ (Γ ⇛ ψ) → Γ ⊢ ψ
completeness : ∀ {I} {Γ : Cxt I} {ϕ : Props} → Γ ⊨ ϕ → Γ ⊢ ϕ
completeness = lemma3 ∘ lemma2 ∘ lemma1
```

3.4.1 Proof of Lemma 1. $\Gamma \vDash \psi \to \emptyset \vDash \Gamma \Rrightarrow \psi$

The idea is to move every propositional formula in the context to the right hand side of the turnstile and combine them with the given proposition ψ to construct a tautology. For example, if Γ contains n propositions which are $\phi_1, \phi_2, ..., \phi_n$, the proposition we will construct is $\phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi$. The reason why $\Gamma \Rrightarrow \psi$ is a tautology is simple: the only situation where this evaluates to false is when each proposition in Γ evaluates to true and ψ evaluates to false. However, this is impossible because we know $\Gamma \vDash \psi$ which means ψ must evaluate to true when everything in Γ evaluates to true.

We first implement a function $\Gamma \Rrightarrow \psi$ by induction on Γ:

```
_⇛_ : ∀ {I} (Γ : Cxt I) (ψ : Props) → Props
∅ ⇛ ψ       = ψ
(Γ • ϕ) ⇛ ψ = Γ ⇛ ϕ ⇒ ψ
```

Then we prove the proposition $\Gamma \Rrightarrow \phi$ we constructed is a tautology:

```
lemma1 : ∀ {I} {Γ : Cxt I} {ψ : Props} → Γ ⊨ ψ → ∅ ⊨ Γ ⇛ ψ
lemma1 {Γ = ∅}     {ψ} ∅⊨ψ ρ [∅]≡true = ∅⊨ψ ρ [∅]≡true
```

$$\begin{aligned}&\mathsf{lemma1}\ \{\Gamma\ =\ \Gamma \bullet \phi\}\ \{\psi\}\ \emptyset{\models}\psi\ \rho\ [\![\emptyset]\!]{\equiv}\mathsf{true}\\&\quad =\ \mathsf{lemma1}\ \{\Gamma\ =\ \Gamma\}\ (\lambda\ \rho\text{'}\ \rightarrow\ \mathsf{oneStep}\ \rho\text{'}\ (\emptyset{\models}\psi\ \rho\text{'}))\ \rho\ \mathsf{refl}\end{aligned}$$

The idea of the proof is that we investigate the form of Γ by pattern matching in Agda. The case when Γ is $\varnothing$ is trivial. If Γ has form $(\Gamma \bullet \phi)$, then we use the helper function

$$\begin{aligned}&\mathsf{oneStep}\ :\ \forall\ \rho\ \rightarrow\ (\llbracket\ \Gamma \bullet \phi\ \rrbracket^c\ \rho\ \equiv\ \mathsf{true}\ \rightarrow\ \llbracket\ \psi\ \rrbracket\ \rho\ \equiv\ \mathsf{true})\\&\quad \rightarrow\ (\llbracket\ \Gamma\ \rrbracket^c\ \rho\ \equiv\ \mathsf{true}\ \rightarrow\ \llbracket\ \phi \Rightarrow \psi\ \rrbracket\ \rho\ \equiv\ \mathsf{true})\end{aligned}$$

to prove that by first pattern matching on the truth value of ϕ and ψ. This enables us to move one proposition from the left hand side to the right hand side of $\models$. By repeating this step, we can move all propositions to the right hand side. lemma1 is calling oneStep as many times as required. In the end, we will get a semantical entailment relation: $\emptyset \models \phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi$ which is $\emptyset \models \Gamma \Rrightarrow \psi$.

3.4.2 Proof of Lemma 2. $\emptyset \models \eta\ \rightarrow\ \emptyset \vdash \eta$

This lemma (weak completeness) is the most difficult part of the proof because η can be an arbitrary proposition and we have to present a concrete proof that we can always construct a proof by using deduction rules regardless of the inner structure of the tautology η.

Given a tautology η which contains n propositional atoms $p_0, p_1, ...p_{n-1}$, there are 2^n different lines of valuations in the truth table of η and η always evaluates to true no matter what valuation it uses. The key idea of proving weak completeness is that we encode each line of valuation in the truth table as a specific context. Then we first give an intermediate proof which says that from any one of these 2^n different contexts we can always derive η. Then we embed this proof in the proof of weak completeness by proving that from an empty context ($\emptyset$) we can construct all of these 2^n contexts by using the law of excluded middle.

To encode the 2^n lines of valuation, we introduce a definition Cxt $[\rho]$ which explains how to translate a truth assignment into a context.

Definition 5. *(Cxt[ρ]): Given a valuation ρ of n propositional atoms named $p_0, p_1, ..., p_{n-1}$, we construct a context based on them. $Cxt[\rho]$ is a context which contains n propositions: $\hat{p}_0, \hat{p}_1, ..., \hat{p}_{n-1}$ where $\hat{p}_i = p_i$ if p_i is assigned true by ρ, otherwise $\hat{p}_i = \sim p_i$.*

The two steps in the proof are as follows:

$$\begin{aligned}&\mathsf{lemma21}\ :\ \forall\ \{\eta\ :\ \mathsf{Props}\}\ \rightarrow\ \emptyset \models \eta\ \rightarrow\ (\forall\ \rho\ \rightarrow\ \mathsf{Cxt}\ [\rho] \leqslant\text{-}\mathsf{refl} \vdash \eta)\\&\mathsf{lemma22}\ :\ \forall\ \{\eta\ :\ \mathsf{Props}\}\ \rightarrow\ (\forall\ \rho\ \rightarrow\ \mathsf{Cxt}\ [\rho] \leqslant\text{-}\mathsf{refl} \vdash \eta)\ \rightarrow\ \emptyset \vdash \eta\\&\mathsf{lemma2}\ =\ \mathsf{lemma22} \circ \mathsf{lemma21}\end{aligned}$$

3.4.2.1 ***Formalising Cxt[ρ] in Agda***

Before formalising $Cxt[\rho]$, we first define $\hat{p}[\rho]$ to construct $\hat{p}_i$ given a valuation ρ and a number x which has type $\mathsf{Fin\ n}$ in Agda in preparation for defining $Cxt[\rho]$.

$$
\begin{aligned}
&\hat{p}_i\ [\rho]\ :\ \forall\ \rho\ \rightarrow\ \mathsf{Fin\ n}\ \rightarrow\ \mathsf{Props}\\
&\hat{p}_i\ [\rho]\ \rho\ \mathsf{x}\ \mathbf{with}\ \rho\ \mathsf{x}\\
&\hat{p}_i\ [\rho]\ \rho\ \mathsf{x}\ \mid\ \mathsf{true}\ =\ \mathsf{patom\ x}\\
&\hat{p}_i\ [\rho]\ \rho\ \mathsf{x}\ \mid\ \mathsf{false}\ =\ \sim \mathsf{patom\ x}
\end{aligned}
$$

$Cxt[\rho]$ consists of $\hat{p}_i$s, but we cannot define it in Agda by doing pattern matching on the number of propositional variables n because it is a parameter of the whole proof. Instead we define cut-off contexts: $\mathsf{Cxt}\ [\rho]\ (\alpha)$ will be the first a elements of $Cxt[\rho]$ where $\alpha\ :\ \mathsf{a} \leqslant \mathsf{n}$. This way we can define $\mathsf{Cxt}\ [\rho]\ (\alpha)$ by induction on a. We also need these cut-off contexts to prove lemmas from Lemma 2.2.2 on.

$$
\begin{aligned}
&\mathsf{Cxt}\ [\_]\ :\ \forall\ \rho\ \{\mathsf{a}\}\ \rightarrow\ \mathsf{a} \leqslant \mathsf{n}\ \rightarrow\ \mathsf{Cxt\ l}\\
&\mathsf{Cxt}\ [\_]\ \_\ \{\mathsf{zero}\}\ \ \mathsf{z}\leqslant\mathsf{n}\ =\ \emptyset\\
&\mathsf{Cxt}\ [\_]\ \rho\ \{\mathsf{suc\ a}\}\ \alpha\ \ \ \ =\ \mathsf{Cxt}\ [\rho]\ (\leqslant\Rightarrow\mathsf{pred}\leqslant\ \_\ \_\ \alpha)\ \bullet\ \hat{p}_i\ [\rho]\ (\mathsf{from}\mathbb{N}\leqslant\ \alpha)
\end{aligned}
$$

Another way to represent a and $\mathsf{a} \leqslant \mathsf{n}$ would be to use an $\mathsf{a}\ :\ \mathsf{Fin}\ (\mathsf{suc\ n})$. However, this would add additional difficulties of converting elements of $\mathsf{Fin\ n}$ into $\mathbb{N}$ and vice versa.

3.4.2.2 ***Proof of lemma 2.1***

Lemma 2.1 says that given $\emptyset \vDash \eta$, for all valuations ρ, $\mathsf{Cxt}\ [\rho] \vdash \eta$. We prove it by proving the following two lemmas mutually, and then using lemma 2.1.1.

$$
\begin{aligned}
&\mathsf{lemma211}\ :\ \forall\ \{\psi\}\ \rho\ \rightarrow\ ([\![\ \psi\ ]\!]\ \rho\ \equiv\ \mathsf{true})\ \rightarrow\ (\mathsf{Cxt}\ [\rho] \leqslant\text{-}\mathsf{refl}) \vdash \psi\\
&\mathsf{lemma212}\ :\ \forall\ \{\psi\}\ \rho\ \rightarrow\ ([\![\ \psi\ ]\!]\ \rho\ \equiv\ \mathsf{false})\ \rightarrow\ (\mathsf{Cxt}\ [\rho] \leqslant\text{-}\mathsf{refl}) \vdash \sim \psi\\
&\mathsf{lemma21}\ \sigma\ \rho\ =\ \mathsf{lemma211}\ \rho\ (\sigma\ \rho\ \mathsf{refl})
\end{aligned}
$$

$\leqslant$-refl is the proof that the relation $\_\leqslant\_$ is reflexive.

We illustrate the proof of $\mathsf{Cxt}\ [\rho] \vdash \psi$ by the case when ψ has form $\phi_1 \wedge \phi_2$ as an example. The other cases are similar.

$$
\begin{aligned}
&\mathsf{lemma211}\ \{\psi\ =\ \phi_1 \wedge \phi_2\}\ \rho\ \_\\
&\quad\mathbf{with}\ [\![\ \phi_1\ ]\!]\ \rho\ \mid\ \mathsf{inspect}\ [\![\ \phi_1\ ]\!]\ \rho\ \mid\ [\![\ \phi_2\ ]\!]\ \rho\ \mid\ \mathsf{inspect}\ [\![\ \phi_2\ ]\!]\ \rho\\
&\mathsf{lemma211}\ \{\psi\ =\ \phi_1 \wedge \phi_2\}\ \rho\ \_\ \mid\ \mathsf{true}\ \ \mid\ [\phi_1{\equiv}\mathsf{true}]\ \mid\ \mathsf{true}\ \ \mid\ [\phi_2{\equiv}\mathsf{true}]\\
&\quad\ =\ \wedge\text{-}\mathsf{i}\ (\mathsf{lemma211}\ \rho\ \phi_1{\equiv}\mathsf{true})\ (\mathsf{lemma211}\ \rho\ \phi_2{\equiv}\mathsf{true})\\
&\mathsf{lemma211}\ \{\psi\ =\ \phi_1 \wedge \phi_2\}\ \rho\ ()\ \mid\ \mathsf{true}\ \ \mid\ [\_]\ \ \ \ \ \ \ \ \mid\ \mathsf{false}\ \mid\ [\_]\\
&\mathsf{lemma211}\ \{\psi\ =\ \phi_1 \wedge \phi_2\}\ \rho\ ()\ \mid\ \mathsf{false}\ \mid\ [\_]\ \ \ \ \ \ \ \ \mid\ \_\ \ \ \ \mid\ [\_]
\end{aligned}
$$

We inspect the meaning of ϕ_1 and ϕ_2, and if both are true, we use the induction hypotheses. If one of them is false, the type of the third argument of

the function which was originally $[\![\ \psi\ ]\!]\ \rho\ \equiv$ true will reduce to false $\equiv$ true and this can be pattern matched with the absurd pattern () of Agda. If there is an absurd pattern on the left hand side, we don't need a right hand side.

3.4.2.3 ***Proof of lemma 2.2***

Lemma 2.2 says that if for all valuations ρ, $Cxt[\rho] \vdash \eta$ is valid, then we can deduce η from only the basic deduction rules without the help of any other premises. In other words, η is a theorem.

```
lemma22 : ∀ {η : Props} → (∀ ρ → Cxt [ρ] ⩽-refl ⊢ η) → ∅ ⊢ η
```

The idea of the proof is straightforward. We start with a context containing n propositions if η has at most n propositional atoms. In the end, the context contains zero propositions. It is natural to think of reducing the number of propositions in the context one at a time, by using the law of excluded middle.

To prove Lemma 2.2, we first prove a more general lemma. As we cannot do induction on n, we need to introduce introduce a new parameter a on which we do induction, and b will remain unchanged during computation.

```
lemma221 : ∀ {η} {a b} {an : a ⩽ n} {bn : b ⩽ n} (ba : b ⩽ a)
           → (∀ ρ → Cxt [ρ] an ⊢ η) → (∀ ρ → Cxt [ρ] bn ⊢ η)
```

One special case of Lemma 2.2.1 is Lemma 2.2 where a = n and b = 0 and we add a dummy valuation:

```
lemma22 σ = lemma221 {bn = z⩽n} z⩽n σ (λ _ → true) .
```

Lemma 2.2.2 will reduce the length of the context by one:

```
lemma222 : ∀ {a} {α : a < n} {η : Props}
           → (∀ ρ → Cxt [ρ] {suc a} α ⊢ η)
           → (∀ ρ → Cxt [ρ] {a} (⩽⇒pred⩽ _ _ α) ⊢ η) .
```

This is the step that is repeated by lemma 2.2.1 until we reach b:

```
lemma221 {a = a} {b} ba σ ρ with a ≟ b
lemma221 ba σ ρ | yes refl
  = subst (λ z → Cxt [ρ] z ⊢ _) (⩽-unique _ _) (σ ρ)
lemma221 {a = zero} z⩽n σ ρ | no ¬p
  = subst (λ x → Cxt [ρ] x ⊢ _) (⩽-unique _ _) (σ ρ)
lemma221 {a = suc a} ba σ ρ | no ¬p
  = lemma221 (lemma-⩽2 ba (¬p ∘ _⁻¹)) (lemma222 σ) ρ
```

subst replaces equal elements in a type (subst : (P : A → Set) → a ≡ b → P a → P b), ⩽-unique says that proofs of the ordering relation are unique, lemma-⩽2 is a helper lemma regarding orderings.

The proof of lemma 2.2.2 is using the excluded middle:

```
lemma222 {a = a} {α} {η} σ ρ
  = ∨-e (lem {ψ = patom (fromℕ⩽ α)})
    (subst (λ z → z ⊢ η) (lemma-Cxt ρ α) (σ (ρ [α ↦ true])))
    (subst (λ z → z ⊢ η) (lemma-Cxt ρ α) (σ (ρ [α ↦ false])))
```

The proof tree is the following:

$$\dfrac{\dfrac{}{\hat{p}_0,\hat{p}_1,...,\hat{p}_{a-1}\vdash p_a\vee\sim p_a}\ \text{lem} \qquad \hat{p}_0,\hat{p}_1,...,\hat{p}_{a-1},p_a\vdash\eta \qquad \hat{p}_0,\hat{p}_1,...,\hat{p}_{a-1},\sim p_a\vdash\eta}{\hat{p}_0,\hat{p}_1,...,\hat{p}_{a-1}\vdash\eta}\ \vee\text{-e}$$

Obviously the first premise can be deduced by using LEM. The key is to get the second and the third premises by induction. First we have a powerful condition in our hands (look at the type of lemma222): $\forall\ \rho, \mathsf{Cxt}\ [\rho]\ (\mathsf{suc\ a}) \vdash \eta$. For a fixed ρ, we know that Cxt $[\rho]$ (a) is $\hat{p}_0, \hat{p}_1, ..., \hat{p}_{a-1}$. We then construct two specific valuations ρ [a ↦ true] and ρ [a ↦ false]. In the former, the first a assignments are equal to the first a assignments in \rho, and the (1 + a)th element is true. Now we can see that Cxt [ρ [a ↦ true]] (suc a) is $\hat{p}_0, \hat{p}_1, ..., \hat{p}_{a-1}, p_a$. As we mentioned above, we have a powerful premise in our hands: $\forall\ \rho\ \to$ Cxt $[\rho]$ (suc a) ⊢ η. We replace the general ρ by our ρ [a ↦ true]. From this we get $\hat{p}_0, \hat{p}_1, ..., \hat{p}_{a-1}, p_a\ \vdash\ \eta$ which is the second premise in the proof tree. Similarly, we construct another valuation ρ [a ↦ false] and get a proof for the third premise in the proof tree above.

lemma-Cxt is a helper lemma stating that Cxt [ρ [a ↦ true]] (1 + a) is the same as the composition of the contexts Cxt $[\rho]$ a and patom a (with some additional noise given by proofs that a is less than n).

3.4.3 Lemma 3 $\emptyset \vdash (\Gamma \Rrightarrow \psi)\ \to\ \Gamma \vdash \psi$

This is the last thing we need to prove. The expanded view of $\Gamma\ \Rrightarrow\ \psi$ is $\phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi$. We present the proof for how to move one ϕ_i from the right hand side to the left hand side. This can be used recursively to move all ϕs to the left.

$$\dfrac{\phi_1, \phi_2, ..., \phi_n \vdash \phi_1 \Rightarrow \phi_2 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi \qquad \dfrac{\phi_1 \in \phi_1, \phi_2, ..., \phi_n}{\phi_1, \phi_2, ..., \phi_n \vdash \phi_1}\ \text{var, weaken}}{\phi_1, \phi_2, ...\phi_n \vdash \phi_2 \Rightarrow \phi_3 \Rightarrow ... \Rightarrow \phi_n \Rightarrow \psi}\ \Rightarrow\text{-e}$$

In Agda, its formalisation is given as follows:

```
lemma3 : ∀ {l} {Γ : Cxt l} {ψ : Props} → ∅ ⊢ (Γ ⇛ ψ) → Γ ⊢ ψ
lemma3 {Γ = ∅} {ψ} σ = σ
lemma3 {Γ = Γ • ϕ} {ψ} σ
  = ⇒-e (weaken (lemma3 {Γ = Γ} σ)) var
```

4 Conclusion

This paper implements in Agda a formalisation of the proof for soundness and completeness of classical propositional logic. In order to construct the proof with Agda recognisable functions, we have to find some more general propositions to prove so that the functions we need to construct in Agda are structurally recursive and can pass the termination checker. The original proof then will become a special case of the more general propositions that we give. This intention in return provides us with a higher level view of what the proof actually says. In the end, we achieve a rigorous and clear proof of completeness theorem in classical propositional logic.

An interesting further step would be to investigate the computational content of this proof in the style of normalisation by completeness ([3], [2]). The completeness and the soundness proofs can be composed which gives a normalisation procedure on proof trees (which can be thought about as computer programs).

```
nbc : ∀ {l} {Γ : Cxt l} {φ : Props} → Γ ⊢ φ → Γ ⊢ φ
nbc = completeness ∘ soundness
```

References

1. The Agda Wiki. (2015), http://wiki.portal.chalmers.se/agda
2. Altenkirch, T., Hofmann, M., Streicher, T.: Categorical reconstruction of a reduction free normalization proof. In: Pitt, D., Rydeheard, D.E., Johnstone, P. (eds.) Category Theory and Computer Science. pp. 182–199. LNCS 953 (1995)
3. Berger, U., Schwichtenberg, H.: An inverse of the evaluation functional for typed λ–calculus. In: Vemuri, R. (ed.) Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science. pp. 203–211. IEEE Computer Society Press, Los Alamitos (1991)
4. Blanchette, J.C., Popescu, A., Traytel, D.: Unified classical logic completeness - a coinductive pearl. In: IJCAR'14. pp. 46–60 (2014)
5. Coquand, C.: A formalised proof of the soundness and completeness of a simply typed lambda-calculus with explicit substitutions. Higher-Order and Symbolic Computation 15(1), 57–90 (2002), http://dx.doi.org/10.1023/A%3A1019964114625
6. van Dalen, D.: Logic and structure (3. ed.). Universitext, Springer (1994)
7. Griffin, T.G.: A formulae-as-types notion of control. In: In Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages. pp. 47–58. ACM Press (1990)
8. Herbelin, H., Lee, G.: Forcing-based cut-elimination for gentzen-style intuitionistic sequent calculus. In: Logic, Language, Information and Computation, 16th International Workshop, WoLLIC 2009, Tokyo, Japan, June 21-24, 2009. Proceedings. pp. 209–217 (2009), http://dx.doi.org/10.1007/978-3-642-02261-6_17
9. Huth, M., Ryan, M.: Logic in Computer Science: modelling and reasoning about systems (Chinese edition). Cambridge University Press (2005)
10. Johnstone, P.T.: Notes on logic and set theory. Cambridge University Press (1987)

11. Norell, U.: Towards a practical programming language based on dependent type theory. Ph.D. thesis, Chalmers University of Technology (2007)
12. Program, T.U.F.: Homotopy type theory: Univalent foundations of mathematics. Tech. rep., Institute for Advanced Study (2013)
13. Troelstra, A., van Dalen, D.: Constructivism in Mathematics. Studies in Logic and the Foundations of Mathematics, Elsevier Science (1988), http://books.google.co.uk/books?id=-tc2qp0-2bsC
14. Urban, C.: Classical logic and computation. Ph.D. thesis, University of Cambridge (2000)
15. Veldman, W.: An intuitiomstic completeness theorem for intuitionistic predicate logic. Journal of Symbolic Logic 41, 159–166 (3 1976), http://journals.cambridge.org/article_S0022481200051859