Abstract

This thesis focuses on formalizing the Lindenbaum-Tarski algebra in Cubical Agda, an area of mathematical logic that involves the quotient of the algebra of formulas by an equivalence relation defined in terms of provability. To achieve this goal, we first formalize classical propositional logic and its properties in Agda, and define the equivalence relation on formulas. We then proceed to define the Lindenbaum-Tarski algebra in Cubical Agda and prove its soundness and properties. In particular, we show that the algebra is a complemented distributive lattice, which implies that it is Boolean.

Contents

1 Introduction **1**

2 Agda proof assistant **1**
2.1 Propositions as types 2
2.2 Simply typed functions and data types 2
2.3 Dependent types 3
2.4 Cubical Agda 4

3 Propositional calculus in Agda **4**
3.1 Formulas 4
3.2 Context 6
3.3 Inference rules 6
3.3.1 Law of excluded middle 7
3.3.2 Logical connectives 7
3.3.3 Structural rules 9
3.4 Properties of a propositional calculus 10
3.4.1 Equivalence relation 10
3.4.2 Commutativity 12
3.4.3 Associativity 13
3.4.4 Distributivity 15

4 Lindenbaum-Tarski algebra in Cubical Agda **19**
4.1 Definition 19
4.2 Binary operations and propositional constants 19
4.3 Proof that the Lindenbaum-Tarski algebra is Boolean 23
4.4 Soundness 28

5 Conclusions **28**
5.1 Possible improvements 28
5.2 Future work 28

Appendix A LindenbaumTarski.agda **30**

1 Introduction

`Agda` is a powerful dependently typed programming language that is based on Martin Löf's type theory. Its strong typing and dependent types make it an ideal proof assistant for formalizing mathematical proofs.[1] Formalizing mathematics in Agda provides several benefits, such as creating machine-checkable proofs that ensure the correctness of a proof and detecting errors or gaps in reasoning that may not be apparent through traditional mathematical proofs. This deepens our understanding and verification of mathematical concepts and proofs. In Section 2, we will provide a brief introduction to `Agda` and types.

The Lindenbaum-Tarski algebra is an area of mathematical logic that involves factoring the algebra of formulas by an equivalence relation defined in terms of provability.[6] The Lindenbaum-Tarski algebra of a context Γ consists of the equivalence classes of formulas in Γ. In this thesis, we aim to formalize the Lindenbaum-Tarski algebra in `Cubical Agda`, which is a variant of `Agda` that is extended with cubical type theory.

The first step towards our goal is to formalize classical propositional logic and its properties in `Agda`, defining the equivalence relation on formulas such that two formulas are equivalent if and only if they are provably equivalent in the context. This will be the focus of Section 3.

In Section 4, we will move on to define the Lindenbaum-Tarski algebra in Cubical Agda, proving soundness and some properties of the algebra. Our primary goal will be to formalize that the algebra is a Boolean algebra. We will use the fact that a Boolean algebra can be viewed as a complemented distributive lattice to formalize that the Lindenbaum-Tarski algebra is Boolean.

This formalization will provide the `agda/cubical` library with a formalization of Lindenbaum-Tarski algebra, that is currently missing, and allow for the verification of mathematical proofs related to Lindenbaum-Tarski algebra in the future.

2 Agda proof assistant

Agda is a powerful dependently typed programming language that is based on Martin Löf's type theory. It allows us to encode mathematical propositions as types and their proofs as programs, ensuring the correctness and consistency of our reasoning. The language's strong type system also provides powerful tools for automatically verifying the correctness of proofs.[2]

Furthermore, Agda is a total language, which means that any program e of type T will always terminate with a value in T. This guarantees the absence of runtime errors and prevents the creation of non-terminating programs, unless explicitly requested by the programmer.[1]

2.1 Propositions as types

Propositions as types is a concept that links logical propositions with types in a programming language. It is based on the idea that a proof of a proposition is analogous to a program that satisfies the type associated with the proposition.

In this context, the introduction and elimination rules for logical connectives can be viewed as operations that construct and deconstruct values of the corresponding types. For instance, the introduction rule for conjunction states that given proofs of two propositions, we can construct a proof of their conjunction by pairing the two proofs together. This can be seen as a function that takes two values of the corresponding types and returns a pair value.

Conversely, the elimination rule for conjunction says that given a proof of a conjunction, we can extract proofs of its two conjuncts by projecting the pair onto each component. This can be viewed as a function that takes a pair value and returns two values of the corresponding types.

This is similar to the concept of product types in programming languages, where a product type represents a pair of values. The introduction form of a product type is a pair, and the elimination forms are projection functions that extract the individual components of the tuple. Table 1 summarizes the correspondence between propositions and types and between proofs and programs.

Prop	Type
$\top$	unit
$\bot$	void
$\phi_1 \wedge \phi_2$	$\tau_1 \times \tau_2$
$\phi_1 \supset \phi_2$	$\tau_1 \to \tau_2$
$\phi_1 \vee \phi_2$	$\tau_1 + \tau_2$

Table 1: Propositions as types

This allows us to reason about logical propositions in terms of programming language types, and to use the tools and techniques of programming languages like `Agda` to reason about logical proofs.

2.2 Simply typed functions and data types

A data declaration introduces a new datatype with its name, type, and constructors, along with their types. For instance, the Boolean type can be defined as follows:

```
data Bool : Type where
  true : Bool
  false : Bool
```

Here, `Bool` is the name of the datatype, and `true` and `false` are its constructors. Functions over the datatype `Bool` can be defined using pattern matching, similar to `Haskell`. For example, the `not` function can be defined as follows:

```
not : Bool → Bool
not true = false
not false = true
```

We start by defining the type of `not` as a function from `Bool` to `Bool` and then we define the function by pattern matching on the arguments. `Agda` ensures that the pattern covers all cases and will not accept a function with missing patterns.

The natural numbers can be defined as an inductively defined type, as follows:

```
data ℕ : Type where
  zero : ℕ
  suc : ℕ → ℕ
```

Here, `zero` represents the natural number 0, and `suc` n represents the successor of the natural number n. We can define addition on natural numbers using a recursive function, as follows:

```
_+_ : ℕ → ℕ → ℕ
zero + m = m
suc n + m = suc (n + m)
```

If a name contains underscores ('_') in the definition, they represent where the arguments go. So in this case we get an infix operator, and we write 'm + n' instead of '+ m n', which would have been the case if the name was just '+'. We can set the precedence of an infix operator with an `infix` declaration:

```
infix 25 _+_
```

Data types can also be parameterized by other types. For instance, the type of lists with elements of an arbitrary type is defined as follows:

```
infix 20 _::_
data List (A : Type) : Type where
  [] : List A
  _::_ : A → List A → List A
```

This declares the datatype `List` as a function of a type A that maps to the type `List` A. It has two constructors, '[ ]' for an empty list and '::' for a list that has an element of type A followed by a list of type `List` A. The underscore before '::' is a placeholder for the first argument of the constructor (an element of type A), and the underscore after '::' is a placeholder for the second argument of the constructor (a list of type `List` A).

2.3 Dependent types

A dependent type is a type that depends on elements of another type. An example of a dependent type is a dependent function, where the result type depends on the value of the argument. In Agda, this is denoted by $(x : A) \to B$,

representing functions that take an argument x of type A and produce a result of type B. A special case is when x itself is a type. For instance, we can define the identity function

```
id : (A : Type) → A → A
id A x = x
```

This function takes a type argument A and an element x of type A, and returns x. In `Agda` it is possible to use implicit arguments. To declare an argument as implicit we use curly braces instead of parenthesis when declaring the type argument. In particular, $\{A : Set\} \to B$ means the same thing as $(A : Set) \to B$, but we don't need to provide the type explicitly, the type checker will try to infer it for us. We can now redefine the identity function above as follows:

```
id' : {A : Type} → A → A
id' x = x
```

Note that we no longer need to supply the type when the function is applied.

2.4 Cubical Agda

In this project, we will be working with quotient types, which requires us to use `Cubical Agda`. `Cubical Agda` is an extension of `Agda` that incorporates features from Cubical Type Theory, which is necessary for dealing with set quotients.[1] However, it is important to note that we will only be using a small portion of the `agda/cubical` library, and any cubical theory is beyond the scope of this thesis.

3 Propositional calculus in Agda

Propositional calculus is a formal system that consists of a set of propositional constants, symbols, inference rules, and axioms. The symbols in propositional calculus represent logical connectives and parentheses, and are used to construct well-formed formulas that follow the syntax of the system. The inference rules specify how these symbols can be used to derive additional statements from the initial assumptions, which are given by the axioms of the system.

The reader is expected to be somewhat familiar with propositional calculus and natural deduction, if not they can refer to [3] or [5] which are the basis for most of the definitions and rules used in this section.

3.1 Formulas

In order to represent propositional calculus in `Agda` we first need to define the type for formulas and the language in which we express them.

Definition 3.1 (Language)**.** *The language $\mathcal{L}$ of propositional calculus consists of*

- *proposition symbols:* $p_0, p_1, \ldots, p_n$,
- *logical connectives:* $\wedge, \vee, \neg, \top, \bot$,
- *auxiliary symbols:* (,).

Note that we have omitted the common logical connectives $\rightarrow$ and $\leftrightarrow$. This is because we can define them using other connectives,

$$\phi \rightarrow \psi \stackrel{\text{def}}{=} \neg\phi \vee \psi,$$
$$\phi \leftrightarrow \psi \stackrel{\text{def}}{=} (\neg\phi \vee \psi) \wedge (\neg\psi \vee \phi),$$

making them redundant. It is possible to choose an even smaller set of connectives [5], but we choose this set as it is convenient.

Definition 3.2 (Well formed formula)**.** *The set of well formed formulas is inductively defined as follows:*

- *any propositional constant* $p_0, p_1, \ldots, p_n$ *is a well formed formula,*
- $\top$ *and* $\bot$ *are well formed formulas,*
- *if* p *is a well formed formula, then so is*

$$\neg p,$$

- *if* p_i *and* p_j *are well formed formulas, then so are*

$$p_i \wedge p_j \quad \text{and} \quad p_i \vee p_j.$$

The formula $\top$ *should be thought of as the proposition that is always true, and the formula* $\bot$ *interpreted as the proposition that is always false.*

Now we are ready to define the type for formulas in `Agda` using the definition above.

Agda definition 1 (Formula)**.**

```
data Formula : Type where
  _∧_ : Formula → Formula → Formula
  _∨_ : Formula → Formula → Formula
  ¬_  : Formula → Formula
  const : ℕ → Formula
  ⊥   : Formula
  ⊤   : Formula
```

3.2 Context

Definition 3.3 (Context)**.** *A context is a set of sentences in the language $\mathcal{L}$. The set is defined inductively as follows:*

- *the empty set is a context*
- *if Γ is a context, then $\Gamma \cup \{\phi\}$ is also a context, where ϕ a formula.*

We will sometimes write Γ, ϕ instead of $\Gamma \cup \{\phi\}$, but they should both be thought of as the latter. In `Agda` we can represent context as a list type.

Agda definition 2 (Context)**.**

```
data ctxt : Type where
  ∅   : ctxt
  _,_ : ctxt → Formula → ctxt
```

We also need a way to determine if a given formula is in a given context.

Definition 3.4. *For all contexts Γ and all formulas ϕ and ψ*

- $\phi \in \Gamma \cup \{\phi\}$,
- *if $\phi \in \Gamma$, then $\phi \in \Gamma \cup \{\psi\}$.*

Agda definition 3 (Lookup)**.**

```
data _∈_ : Formula → ctxt → Type where
  Z : ∀ {Γ ϕ} → ϕ ∈ Γ , ϕ
  S : ∀ {Γ ϕ ψ} → ϕ ∈ Γ → ϕ ∈ Γ , ψ
```

3.3 Inference rules

Inference rules are used to derive new propositions from existing ones. In order to formalize the process of proving propositions in our system, we introduce a new data type for provability. This data type has as its inhabitants all of the possible inference rules that can be used to prove a given proposition.

Agda definition 4.

```
data _⊢_ : ctxt → Formula → Type where
```

⋮

3.3.1 Law of excluded middle

Definition 3.5. *The law of excluded middle states that for every proposition, either the proposition or its negation is true.*

This principle is equivalent to the statement that $\phi \vee \neg\phi$ is a tautology, where ϕ is any proposition. Given the above definition, we can introduce an inference rule for LEM. This rule requires no premise, as LEM is a tautology.

$$\frac{}{\Gamma \vdash \phi \vee \neg\phi}\ \text{LEM}$$

Agda definition 5 (Law of excluded middle)**.**

```
LEM : {Γ : ctxt} {ϕ : Formula}
    → Γ ⊢ ϕ ∨ ¬ ϕ
```

By using the LEM inference rule in our system, we can derive new propositions and prove theorems based on the principle of excluded middle.

3.3.2 Logical connectives

Rules for the logical connectives come in pairs of introduction and elimination rules, apart from $\top$, which has only an introduction rule, and $\bot$, which has only an elimination rule.

The introduction rule for conjunction states that if there is a derivation of ϕ from Γ, and a derivation of ψ from Γ, then we can conclude that there is a derivation of $\phi \wedge \psi$ from Γ.

$$\frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}\ \wedge\text{-I}$$

Agda definition 6 (Conjunction introduction)**.**

```
∧-I : {Γ : ctxt} {ϕ ψ : Formula}
    → Γ ⊢ ϕ
    → Γ ⊢ ψ
    → Γ ⊢ ϕ ∧ ψ
```

The corresponding elimination rules says that if there is some derivation concluding in $\phi \wedge \psi$ from Γ, then we can conclude that there is a derivation of ϕ, and a derivation of ψ, from Γ.

$$\frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi}\ \wedge\text{-E}_1 \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}\ \wedge\text{-E}_2$$

Agda definition 7 (Conjunction elimination)**.**

```
∧-E₁ : {Γ : ctxt} {ϕ ψ : Formula}
     → Γ ⊢ ϕ ∧ ψ
     → Γ ⊢ ϕ

∧-E₂ : {Γ : ctxt} {ϕ ψ : Formula}
     → Γ ⊢ ϕ ∧ ψ
     → Γ ⊢ ψ
```

For disjunction, we have two introduction rules. If we can derive a formula ϕ from Γ, then we can also derive $\phi \vee \psi$ from Γ. Similarly, if we can derive ψ from Γ, then we can also derive $\phi \vee \psi$ from Γ.

$$\frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}\ \vee\text{-I}_1 \qquad \frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi}\ \vee\text{-I}_2$$

Agda definition 8 (Disjunction introduction)**.**

```
∨-I₁ : {Γ : ctxt} {ϕ ψ : Formula}
     → Γ ⊢ ψ
     → Γ ⊢ ϕ ∨ ψ

∨-I₂ : {Γ : ctxt} {ϕ ψ : Formula}
     → Γ ⊢ ϕ
     → Γ ⊢ ϕ ∨ ψ
```

The elimination rule for disjunction is a bit more involved. If $\phi \vee \psi$ can be deduced from Γ, then we can conclude $\Gamma \vdash \gamma$ if the extended contexts Γ, ϕ and Γ, ψ both conclude in γ.

$$\frac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \gamma \qquad \Gamma, \psi \vdash \gamma}{\Gamma \vdash \gamma}\ \vee\text{-E}$$

Agda definition 9 (Disjunction elimination)**.**

```
∨-E : {Γ : ctxt} {ϕ ψ γ : Formula}
    → Γ ⊢ ϕ ∨ ψ
    → Γ : ϕ ⊢ γ
    → Γ : ψ ⊢ γ
    → Γ ⊢ γ
```

Definition 3.6. *A context Γ is inconsistent if $\Gamma \vdash \phi$ and $\Gamma \vdash \neg\phi$, or equivalently $\Gamma \vdash \bot$. A context that is not inconsistent, is called consistent.*

The definition of inconsistency and the law of excluded middle together motivates the introduction and elimination rules for negation.

$$\frac{\Gamma, \phi \vdash \bot}{\Gamma \vdash \neg\phi}\ \neg\text{-I} \qquad \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \neg\phi}{\Gamma \vdash \bot}\ \neg\text{-E}$$

Agda definition 10 (Negation introduction)**.**

```
¬-I : {Γ : ctxt} {ϕ : Formula}
  → Γ : ϕ ⊢ ⊥
  → Γ ⊢ ¬ ϕ
```

Agda definition 11 (Negation elimination)**.**

```
¬-E : {Γ : ctxt} {ϕ : Formula}
  → Γ ⊢ ϕ
  → Γ ⊢ ¬ ϕ
  → Γ ⊢ ⊥
```

In Definition 3.2 we mentioned that $\top$ should be thought of as the proposition that is always true. In particular, it is a tautology and as such trivially true in all contexts, and should be introduced with no premise.

$$\frac{}{\Gamma \vdash \top}\ \top\text{-I}$$

Agda definition 12 ($\top$ introduction)**.**

```
⊤-I : {Γ : ctxt} → Γ ⊢ ⊤
```

If a context is inconsistent one can derive anything from it, which leads to the elimination rule for $\bot$.

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \phi}\ \bot\text{-E}$$

Agda definition 13 ($\bot$ elimination)**.**

```
⊥-E : {Γ : ctxt} {ϕ : Formula}
  → Γ ⊢ ⊥
  → Γ ⊢ ϕ
```

3.3.3 Structural rules

Weakening is a structural rule that states that we can extend the hypothesis with additional members.

$$\frac{\Gamma \vdash \phi}{\Gamma, \psi \vdash \phi}\ \text{WEAKENING}$$

Agda definition 14 (Weakening)**.**

```
weakening : {Γ : ctxt} {ϕ ψ : Formula}
          → Γ ⊢ ψ
          → Γ : ϕ ⊢ ψ
```

The structural rule of exchange allows us to permute the formulas in the context. However, we will be using a stricter version of exchange, where we can only permute the two formulas at the end of the context. This version is easier to implement and still satisfies our needs.

$$\frac{\Gamma, \phi, \psi \vdash \gamma}{\Gamma, \psi, \phi \vdash \gamma}\ \text{EXCHANGE}$$

Agda definition 15 (Exchange)**.**

```
exchange : {Γ : ctxt} {ϕ ψ γ : Formula}
         → (Γ : ϕ) : ψ ⊢ γ
         → (Γ : ψ) : ϕ ⊢ γ
```

3.4 Properties of a propositional calculus

In this section we aim to prove some properties of propositional calculus as they will be important later.

3.4.1 Equivalence relation

Definition 3.7. *Let S be the set of all the sentences of $\mathcal{L}$. Define the relation $\sim$ such that for $\phi, \psi \in S$,*

$$\phi \sim \psi \qquad \textit{iff} \qquad \Gamma, \phi \vdash \psi \textit{ and } \Gamma, \psi \vdash \phi$$

Because the definition of the relation is a pair it is natural to define it in `Agda` as a product type.

Agda definition 16 (Equivalence relation)**.**

```
_∼_ : Formula → Formula → Type
ϕ ∼ ψ = Γ : ϕ ⊢ ψ × Γ : ψ ⊢ ϕ
```

Before we prove that this is an equivalence relation we will prove a useful lemma.

Lemma 3.1. *Given $\Gamma, \phi \vdash \psi$ and $\Gamma, \psi \vdash \gamma$, it follows that $\Gamma, \phi \vdash \gamma$*

Proof. This is done through natural deduction using the deduction rules defined previously.

$$\dfrac{\dfrac{\Gamma, \phi \vdash \psi}{\Gamma, \phi \vdash \psi \vee \gamma}\ \vee\text{-I}_2 \qquad \dfrac{\dfrac{\Gamma, \psi \vdash \gamma}{\Gamma, \psi, \phi \vdash \gamma}\ \text{WEAKENING}}{\Gamma, \phi, \psi \vdash \gamma}\ \text{EXCHANGE} \qquad \dfrac{\gamma \in \Gamma, \phi, \gamma}{\Gamma, \phi, \gamma \vdash \gamma}\ \text{AXIOM}}{\Gamma, \phi \vdash \gamma}\ \vee\text{-E}$$

□

Agda proof.

```
⊢trans : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ⊢ γ → Γ : γ ⊢ ψ → Γ : ϕ ⊢ ψ
⊢trans A B = ∨-E (∨-I₂ A) (exchange (weakening B)) (axiom Z)
```

□

Theorem 3.1. *The relation $\sim$ is an equivalence relation.*

Proof. It follows immediately from the axiom rule,

$$\dfrac{\phi \in \Gamma, \phi}{\Gamma, \phi \vdash \phi}\ \text{AXIOM}$$

that $\varphi \sim \varphi$, so the relation is reflexive.

It should be clear that $\Gamma, \phi \vdash \psi$ and $\Gamma, \psi \vdash \phi$ is just a pair of proofs, hence it does not matter in which order we give them. This means that the relation is also symmetric.

In order to prove transitivity we need to show that, given $\phi \sim \gamma$ and $\gamma \sim \psi$, it holds that $\phi \sim \psi$. By Definition 3.7 we have the following:

(i) $\Gamma, \phi \vdash \gamma$,

(ii) $\Gamma, \gamma \vdash \phi$,

(iii) $\Gamma, \gamma \vdash \psi$,

(iv) $\Gamma, \psi \vdash \gamma$.

Now we can apply Lemma 3.1 on (i) and (iii) to conclude $\Gamma, \phi \vdash \psi$, and again to (iv) and (ii) to conclude $\Gamma, \psi \vdash \phi$. Thus we have shown that $\phi \sim \psi$.

□

Agda proof.

```
∼-refl : ∀ (ϕ : Formula) → ϕ ∼ ϕ
∼-refl _ = axiom Z , (axiom Z)


∼-sym : ∀ {ϕ ψ : Formula} → ϕ ∼ ψ → ψ ∼ ϕ
∼-sym (A , B) = B , A
```

```
∼-trans : ∀ {ϕ ψ γ : Formula} → ϕ ∼ γ → γ ∼ ψ → ϕ ∼ ψ
∼-trans x y = ⊢trans (proj₁ x) (proj₁ y) , ⊢trans (proj₂ y) (proj₂ x)
```

□

3.4.2 Commutativity

Proposition 1. *Conjunction is commutative, that is* $\phi \wedge \psi \sim \psi \wedge \phi$.

Proof. We need to show $\Gamma, \phi \wedge \psi \vdash \psi \wedge \phi$ and $\Gamma, \psi \wedge \phi \vdash \phi \wedge \psi$. To show $\Gamma, \phi \wedge \psi \vdash \psi \wedge \phi$, we use natural deduction:

$$\dfrac{\dfrac{\dfrac{\phi \wedge \psi \in \Gamma, \phi \wedge \psi}{\Gamma, \phi \wedge \psi \vdash \phi \wedge \psi}\,\text{AXIOM}}{\Gamma, \phi \wedge \psi \vdash \psi}\,\wedge\text{-E}_2 \qquad \dfrac{\dfrac{\phi \wedge \psi \in \Gamma, \phi \wedge \psi}{\Gamma, \phi \wedge \psi \vdash \phi \wedge \psi}\,\text{AXIOM}}{\Gamma, \phi \wedge \psi \vdash \phi}\,\wedge\text{-E}_1}{\Gamma \vdash \psi \wedge \phi}\,\wedge\text{-I}$$

The proof of the second part is identical up to renaming of the formulas. Together they prove the equivalence $\phi \wedge \psi \sim \psi \wedge \phi$. □

Agda proof.

```
∧-comm : ∀ {ϕ ψ : Formula} → Γ : ϕ ∧ ψ ⊢ ψ ∧ ϕ
∧-comm = ∧-I (∧-E₂ (axiom Z)) (∧-E₁ (axiom Z))


∼-comm-∧ : ∀ {ϕ ψ : Formula} → ϕ ∧ ψ ∼ ψ ∧ ϕ
∼-comm-∧ = ∧-comm , ∧-comm
```

□

Proposition 2. *Disjunction is commutative, that is* $\phi \vee \psi \sim \psi \vee \phi$.

Proof. We need to show $\Gamma, \phi \vee \psi \vdash \psi \vee \phi$ and $\Gamma, \psi \vee \phi \vdash \phi \vee \psi$. Up to renaming of the formulas, the proofs are identical, so it suffices to show $\Gamma, \phi \vee \psi \vdash \psi \vee \phi$. Let $\Gamma' = \Gamma \cup \{\phi \vee \psi\}$, then by natural deduction we have:

$$\dfrac{\dfrac{\phi \vee \psi \in \Gamma'}{\Gamma' \vdash \phi \vee \psi}\,\text{AXIOM} \qquad \dfrac{\dfrac{\phi \in \Gamma', \phi}{\Gamma', \phi \vdash \phi}\,\text{AXIOM}}{\Gamma', \phi \vdash \psi \vee \phi}\,\vee\text{-I}_1 \qquad \dfrac{\dfrac{\psi \in \Gamma', \psi}{\Gamma', \psi \vdash \psi}\,\text{AXIOM}}{\Gamma', \psi \vdash \psi \vee \phi}\,\vee\text{-I}_2}{\Gamma' \vdash \psi \vee \phi}\,\vee\text{-E}$$

Thus we have shown $\Gamma, \phi \vee \psi \vdash \psi \vee \phi$ and $\Gamma, \psi \vee \phi \vdash \phi \vee \psi$, so we have the equivalence $\phi \vee \psi \sim \psi \vee \phi$.

□

Agda proof.

```
∨-comm : {ϕ ψ : Formula} → Γ : ϕ ∨ ψ ⊢ ψ ∨ ϕ
∨-comm = ∨-E (axiom Z) (∨-I₁ (axiom Z)) (∨-I₂ (axiom Z))


∼-comm-∨ : ∀ {ϕ ψ : Formula} → ϕ ∨ ψ ∼ ψ ∨ ϕ
∼-comm-∨ = ∨-comm , ∨-comm
```

□

3.4.3 Associativity

Proposition 3. *Conjunction is associative, that is* $\phi \wedge (\psi \wedge \gamma) \sim (\phi \wedge \psi) \wedge \gamma$.

Proof. We need to show $\Gamma, \phi \wedge (\psi \wedge \gamma) \vdash (\phi \wedge \psi) \wedge \gamma$ and $\Gamma, (\phi \wedge \psi) \wedge \gamma \vdash \phi \wedge (\psi \wedge \gamma)$. Using natural deduction, we can prove both statements. Since the two proofs become identical when using the commutativity property and relabeling the formulas, it suffices to show the first one.

Let $\Gamma' = \Gamma \cup \{\phi \wedge (\psi \wedge \gamma)\}$. Then we can make a deduction $\mathcal{D}_1$ concluding in $\Gamma' \vdash \varphi \wedge \psi$,

$$\dfrac{\dfrac{\dfrac{\dfrac{\phi \wedge (\psi \wedge \gamma) \in \Gamma'}{\Gamma' \vdash \phi \wedge (\psi \wedge \gamma)}\,\text{AXIOM}}{\Gamma' \vdash \psi \wedge \gamma}\,\wedge\text{-E}_2}{\Gamma' \vdash \psi}\,\wedge\text{-E}_1 \qquad \dfrac{\dfrac{\phi \wedge (\psi \wedge \gamma) \in \Gamma'}{\Gamma' \vdash \phi \wedge (\psi \wedge \gamma)}\,\text{AXIOM}}{\Gamma' \vdash \phi}\,\wedge\text{-E}_1}{\Gamma' \vdash \phi \wedge \psi}\,\wedge\text{-I}$$

and another deduction $\mathcal{D}_2$ concluding in $\Gamma' \vdash \gamma$,

$$\dfrac{\dfrac{\dfrac{\phi \wedge (\psi \wedge \gamma) \in \Gamma'}{\Gamma' \vdash \phi \wedge (\psi \wedge \gamma)}\,\text{AXIOM}}{\Gamma' \vdash \psi \wedge \gamma}\,\wedge\text{-E}_2}{\Gamma' \vdash \gamma}\,\wedge\text{-E}_2$$

Now, with the derivations $\mathcal{D}_1$ and $\mathcal{D}_2$ we can, using conjunction introduction, conclude that $\Gamma' \vdash (\phi \wedge \psi) \wedge \gamma$,

$$\dfrac{\mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma' \vdash (\phi \wedge \psi) \wedge \gamma}\,\wedge\text{-I}$$

Thus we have shown that $\Gamma, \phi \wedge (\psi \wedge \gamma) \vdash (\phi \wedge \psi) \wedge \gamma$. Together with the omitted proof we get the equivalence $\phi \wedge (\psi \wedge \gamma) \sim (\phi \wedge \psi) \wedge \gamma$. □

Agda proof.

```
∧-ass1 : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ∧ (ψ ∧ γ) ⊢ (ϕ ∧ ψ) ∧ γ
∧-ass1 = ∧-I (∧-I (∧-E₁ (axiom Z)) (∧-E₁ (∧-E₂ (axiom Z))))
             (∧-E₂ (∧-E₂ (axiom Z)))


∧-ass2 : ∀ {ϕ ψ γ : Formula} → Γ : (ϕ ∧ ψ) ∧ γ ⊢ ϕ ∧ (ψ ∧ γ)
∧-ass2 = ∧-I (∧-E₁ (∧-E₁ (axiom Z)))
             (∧-I (∧-E₂ (∧-E₁ (axiom Z)))
                  (∧-E₂ (axiom Z)))


∼-ass-∧ : ∀ {ϕ ψ γ : Formula} → ϕ ∧ (ψ ∧ γ) ∼ (ϕ ∧ ψ) ∧ γ
∼-ass-∧ = ∧-ass1 , ∧-ass2
```

□

Proposition 4. *Disjunction is associative, that is* $\phi \vee (\psi \vee \gamma) \sim (\phi \vee \psi) \vee \gamma$.

Proof. We need to show $\Gamma, \phi \vee (\psi \vee \gamma) \vdash (\phi \vee \psi) \vee \gamma$ and $\Gamma, (\phi \vee \psi) \vee \gamma \vdash \phi \vee (\psi \vee \gamma)$. Proof is done through natural deduction. Let $\Gamma' = \Gamma \cup \{\phi \vee (\psi \vee \gamma)\}$, then

$$\dfrac{\dfrac{\phi \vee (\psi \vee \gamma) \in \Gamma'}{\Gamma' \vdash \phi \vee (\psi \vee \gamma)}\,\text{AXIOM} \qquad \mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma' \vdash (\phi \vee \psi) \vee \gamma}\,\vee\text{-E}$$

Here we need $\mathcal{D}_1$ to be a deduction of $\Gamma', \phi \vdash (\phi \vee \psi) \vee \gamma$ and and $\mathcal{D}_2$ a deduction of $\Gamma', \psi \vee \gamma \vdash (\phi \vee \psi) \vee \gamma$. We can construct $\mathcal{D}_1$ as follows:

$$\dfrac{\dfrac{\dfrac{\phi \in \Gamma', \phi}{\Gamma', \phi \vdash \phi}\,\text{AXIOM}}{\Gamma', \phi \vdash \phi \vee \psi}\,\vee\text{-I}_2}{\Gamma', \phi \vdash (\phi \vee \psi) \vee \gamma}\,\vee\text{-I}_2$$

Now, let $\Gamma'' = \Gamma' \cup \{\psi \vee \gamma\}$, then $\mathcal{D}_2$ is constructed as follows:

$$\dfrac{\dfrac{\psi \vee \gamma \in \Gamma''}{\Gamma'' \vdash \psi \vee \gamma}\,\text{AXIOM} \quad \dfrac{\dfrac{\psi \in \Gamma'', \psi}{\Gamma'', \psi \vdash \psi}\,\text{AXIOM}}{\Gamma'', \psi \vdash \phi \vee \psi}\,\vee\text{-I}_1}{\Gamma'', \psi \vdash (\phi \vee \psi) \vee \gamma}\,\vee\text{-I}_2 \quad \dfrac{\dfrac{\gamma \in \Gamma'', \gamma}{\Gamma'', \gamma \vdash \gamma}\,\text{AXIOM}}{\Gamma'', \gamma \vdash (\phi \vee \psi) \vee \gamma}\,\vee\text{-I}_1}{\Gamma'' \vdash (\phi \vee \psi) \vee \gamma}\,\vee\text{-E}$$

The proof of $\Gamma, (\phi \vee \psi) \vee \gamma \vdash \phi \vee (\psi \vee \gamma)$ is identical when relabeling the formulas and using that disjunction is commutative as proven earlier. With these results we get the equivalence $\phi \vee (\psi \vee \gamma) \sim (\phi \vee \psi) \vee \gamma$.

□

Agda proof.

```
∨-ass1 : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ∨ (ψ ∨ γ) ⊢ (ϕ ∨ ψ) ∨ γ
∨-ass1 = ∨-E (axiom Z)
             (∨-I₂ (∨-I₂ (axiom Z)))
             (∨-E (axiom Z)
                  (∨-I₂ (∨-I₁ (axiom Z)))
                  (∨-I₁ (axiom Z)))


∨-ass2 : ∀ {ϕ ψ γ : Formula} → Γ : (ϕ ∨ ψ) ∨ γ ⊢ ϕ ∨ (ψ ∨ γ)
∨-ass2 = ∨-E (axiom Z)
             (∨-E (axiom Z)
                  (∨-I₂ (axiom Z))
                  (∨-I₁ (∨-I₂ (axiom Z))))
             (∨-I₁ (∨-I₁ (axiom Z)))


∼-ass-∨ : ∀ {ϕ ψ γ : Formula} → ϕ ∨ (ψ ∨ γ) ∼ (ϕ ∨ ψ) ∨ γ
∼-ass-∨ = ∨-ass1 , ∨-ass2
```

□

3.4.4 Distributivity

Proposition 5. *Distribution over conjunction holds, that is the equivalence* $\phi \wedge (\psi \vee \gamma) \sim (\phi \wedge \psi) \vee (\phi \wedge \gamma)$ *holds.*

Proof. First we show that $\Gamma, \phi \wedge (\psi \vee \gamma) \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)$. Let $\Gamma_1 = \Gamma \cup \{\phi \wedge (\psi \vee \gamma)\}$, then

$$\dfrac{\dfrac{\dfrac{\phi \wedge (\psi \vee \gamma) \in \Gamma_1}{\Gamma_1 \vdash \phi \wedge (\psi \vee \gamma)}\,\text{AXIOM}}{\Gamma_1 \vdash \psi \vee \gamma}\,\wedge\text{-E}_2 \qquad \mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma_1 \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)}\,\vee\text{-E}$$

where $\mathcal{D}_1$ is a derivation concluding in $\Gamma_1, \psi \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)$ and $\mathcal{D}_2$ a derivation concluding in $\Gamma_1, \gamma \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)$. We start by performing the derivation $\mathcal{D}_1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\phi \wedge (\psi \vee \gamma) \in \Gamma_1}{\Gamma_1 \vdash \phi \wedge (\psi \vee \gamma)}\,\text{AXIOM}}{\Gamma_1 \vdash \phi}\,\wedge\text{-E}_1}{\Gamma_1, \psi \vdash \phi}\,\text{WEAKENING} \qquad \dfrac{\psi \in \Gamma_1, \psi}{\Gamma_1, \psi \vdash \psi}\,\text{AXIOM}}{\Gamma_1, \psi \vdash \phi \wedge \psi}\,\wedge\text{-I}}{\Gamma_1, \psi \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)}\,\vee\text{-I}_2$$

Then we derive $\mathcal{D}_2$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\phi \wedge (\psi \vee \gamma) \in \Gamma_1}{\Gamma_1 \vdash \phi \wedge (\psi \vee \gamma)}\,\text{AXIOM}}{\Gamma_1 \vdash \phi}\,\wedge\text{-E}_1}{\Gamma_1, \gamma \vdash \phi}\,\text{WEAKENING} \qquad \dfrac{\gamma \in \Gamma_1, \gamma}{\Gamma_1, \gamma \vdash \gamma}\,\text{AXIOM}}{\Gamma_1, \gamma \vdash \phi \wedge \gamma}\,\wedge\text{-I}}{\Gamma_1, \gamma \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)}\,\vee\text{-I}_1$$

This proves the first part of the equivalence. Now let $\Gamma_2 = \Gamma \cup \{(\phi \wedge \psi) \vee (\phi \wedge \gamma)\}$, then we want to show $\Gamma_2 \vdash \phi \wedge (\psi \vee \gamma)$. This means that if we can find derivations $\mathcal{D}_1'$ concluding in $\Gamma_2 \vdash \phi$ and $\mathcal{D}_2'$ concluding in $\Gamma_2 \vdash (\psi \vee \gamma)$ we can apply conjunction introduction to get the desired result.

We will start with the derivation tree $\mathcal{D}_1'$:

$$\dfrac{\mathcal{D}_1''}{\dfrac{\dfrac{\phi \wedge \psi \in \Gamma_2, \phi \wedge \psi}{\Gamma_2, \phi \wedge \psi \vdash \phi \wedge \psi}\,\text{AXIOM}}{\Gamma_2, \phi \wedge \psi \vdash \phi}\,\wedge\text{-E}_1} \qquad \dfrac{\mathcal{D}_1'''}{\dfrac{\dfrac{\phi \wedge \gamma \in \Gamma_2, \phi \wedge \gamma}{\Gamma_2, \phi \wedge \gamma \vdash \phi \wedge \gamma}\,\text{AXIOM}}{\Gamma_2, \phi \wedge \gamma \vdash \phi}\,\wedge\text{-E}_1}$$

$$\dfrac{\mathcal{D}_1'}{\dfrac{\dfrac{(\phi \wedge \psi) \vee (\phi \wedge \gamma) \in \Gamma_2}{\Gamma_2 \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)}\,\text{AXIOM} \qquad \mathcal{D}_1'' \qquad \mathcal{D}_1'''}{\Gamma_2 \vdash \phi}\,\vee\text{-E}}$$

Next we construct the derivation tree $\mathcal{D}_2'$ as follows:

$$\dfrac{\mathcal{D}_2''}{\dfrac{\dfrac{\dfrac{\phi \wedge \psi \in \Gamma_2, \phi \wedge \psi}{\Gamma_2, \phi \wedge \psi \vdash \phi \wedge \psi}\,\text{AXIOM}}{\Gamma_2, \phi \wedge \psi \vdash \psi}\,\wedge\text{-E}_2}{\Gamma_2, \phi \wedge \psi \vdash \psi \vee \gamma}\,\vee\text{-I}_2} \qquad \dfrac{\mathcal{D}_2'''}{\dfrac{\dfrac{\dfrac{\phi \wedge \gamma \in \Gamma_2, \phi \wedge \gamma}{\Gamma_2, \phi \wedge \gamma \vdash \phi \wedge \gamma}\,\text{AXIOM}}{\Gamma_2, \phi \wedge \gamma \vdash \gamma}\,\wedge\text{-E}_2}{\Gamma_2, \phi \wedge \gamma \vdash \psi \vee \gamma}\,\vee\text{-I}_1}$$

$$\dfrac{\mathcal{D}_2'}{\dfrac{\dfrac{(\phi \wedge \psi) \vee (\phi \wedge \gamma) \in \Gamma_2}{\Gamma_2 \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)}\,\text{AXIOM} \qquad \mathcal{D}_2'' \qquad \mathcal{D}_2'''}{\Gamma_2 \vdash \psi \vee \gamma}\,\vee\text{-E}}$$

Lastly we put them together with conjunction introduction to get the desired result.

$$\dfrac{\mathcal{D}_1' \qquad \mathcal{D}_2'}{\Gamma_2 \vdash \phi \wedge (\psi \vee \gamma)}\,\wedge\text{-I}$$

This concludes the proof that $\phi \wedge (\psi \vee \gamma) \sim (\phi \wedge \psi) \vee (\phi \wedge \gamma)$. □

Agda proof.

```
∧-dist1 : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ∧ (ψ ∨ γ) ⊢ (ϕ ∧ ψ) ∨ (ϕ ∧ γ)
∧-dist1 = ∨-E (∧-E₂ (axiom Z))
              (∨-I₂ (∧-I (weakening (∧-E₁ (axiom Z))) (axiom Z)))
              (∨-I₁ (∧-I (weakening (∧-E₁ (axiom Z))) (axiom Z)))

∧-dist2 : ∀ {ϕ ψ γ : Formula} → Γ : (ϕ ∧ ψ) ∨ (ϕ ∧ γ) ⊢ ϕ ∧ (ψ ∨ γ)
∧-dist2 = ∧-I (∨-E (axiom Z)
                   (∧-E₁ (axiom Z))
                   (∧-E₁ (axiom Z)))
              (∨-E (axiom Z)
                   (∨-I₂ (∧-E₂ (axiom Z)))
                   (∨-I₁ (∧-E₂ (axiom Z))))

∼-dist-∧ : ∀ {ϕ ψ γ : Formula} → ϕ ∧ (ψ ∨ γ) ∼ (ϕ ∧ ψ) ∨ (ϕ ∧ γ)
∼-dist-∧ = ∧-dist1 , ∧-dist2
```

□

Proposition 6. *Distribution over disjunction holds, that is the equivalence* $\phi \vee (\psi \wedge \gamma) \sim (\phi \vee \psi) \wedge (\phi \vee \gamma)$ *holds.*

Proof. We aim to show $\Gamma, \phi \vee (\psi \wedge \gamma) \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)$ and $\Gamma, (\phi \vee \psi) \wedge (\phi \vee \gamma) \vdash \phi \vee (\psi \wedge \gamma)$.

Let $\Gamma_1 = \Gamma \cup \{\phi \vee (\psi \wedge \gamma)\}$. We want to use disjunction elimination here, so we want to find derivations $\mathcal{D}_1$ and $\mathcal{D}_2$, such that

$$\dfrac{\dfrac{\phi \vee (\psi \wedge \gamma) \in \Gamma_1}{\Gamma_1 \vdash \phi \vee (\psi \wedge \gamma)}\,\text{AXIOM} \qquad \mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma_1 \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\,\vee\text{-E}$$

This means $\mathcal{D}_1$ should conclude with $\Gamma_1, \phi \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)$. We construct $\mathcal{D}_1$ as follows:

$$\dfrac{\dfrac{\dfrac{\phi \in \Gamma_1, \phi}{\Gamma_1, \phi \vdash \phi}\,\text{AXIOM}}{\Gamma_1, \phi \vdash \phi \vee \psi}\,\vee\text{-I}_2 \qquad \dfrac{\dfrac{\phi \in \Gamma_1, \phi}{\Gamma_1, \phi \vdash \phi}\,\text{AXIOM}}{\Gamma_1, \phi \vdash (\phi \vee \gamma)}\,\vee\text{-I}_2}{\Gamma_1, \phi \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\,\wedge\text{-I}$$

Then we need to construct $\mathcal{D}_2$, concluding in $\Gamma_1, \psi \wedge \gamma \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)$. This can be done as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\psi \wedge \gamma \in \Gamma_1, \psi \wedge \gamma}{\Gamma_1, \psi \wedge \gamma \vdash \psi \wedge \gamma}\,\text{AXIOM}}{\Gamma_1, \psi \wedge \gamma \vdash \psi}\,\wedge\text{-E}_1}{\Gamma_1, \psi \wedge \gamma \vdash (\phi \vee \psi)}\,\vee\text{-I}_1 \qquad \dfrac{\dfrac{\dfrac{\psi \wedge \gamma \in \Gamma_1, \psi \wedge \gamma}{\Gamma_1, \psi \wedge \gamma \vdash \psi \wedge \gamma}\,\text{AXIOM}}{\Gamma_1, \psi \wedge \gamma \vdash \gamma}\,\wedge\text{-E}_2}{\Gamma_1, \psi \wedge \gamma \vdash (\phi \vee \gamma)}\,\vee\text{-I}_1}{\Gamma_1, \psi \wedge \gamma \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\,\wedge\text{-I}$$

This proves $\Gamma, \phi \vee (\psi \wedge \gamma) \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)$. To prove the second part we let $\Gamma_2 = \Gamma \cup \{(\phi \vee \psi) \wedge (\phi \vee \gamma)\}$, and then we find $\mathcal{D}_1'$ and $\mathcal{D}_2'$ such that

$$\dfrac{\dfrac{\dfrac{(\phi \vee \psi) \wedge (\phi \vee \gamma) \in \Gamma_2}{\Gamma_2 \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\,\text{AXIOM}}{\Gamma_2 \vdash \phi \vee \psi}\,\wedge\text{-E}_1 \qquad \mathcal{D}_1' \qquad \mathcal{D}_2'}{\Gamma_2 \vdash \phi \vee (\psi \wedge \gamma)}\,\vee\text{-E}$$

The derivation $\mathcal{D}_1'$ should conclude in $\Gamma_2, \phi \vdash \phi \vee (\psi \wedge \gamma)$. This is done as follows:

$$\dfrac{\dfrac{\phi \in \Gamma_2, \phi}{\Gamma_2, \phi \vdash \phi}\,\text{AXIOM}}{\Gamma_2, \phi \vdash \phi \vee (\psi \wedge \gamma)}\,\vee\text{-I}_2$$

To derive $\mathcal{D}_2'$, concluding in $\Gamma_2, \psi \vdash \phi \vee (\psi \wedge \gamma)$, we need to use disjunction elimination again. In order to do that we need to find derivations $\mathcal{D}_2''$, concluding in $\Gamma_2, \psi, \phi \vdash \phi \vee (\psi \wedge \gamma)$, and $\mathcal{D}_2'''$, concluding in $\Gamma_2, \psi, \gamma \vdash \phi \vee (\psi \wedge \gamma)$, such that this derivation $\mathcal{D}_2'$ holds.

$$\dfrac{\dfrac{\dfrac{\dfrac{(\phi \vee \psi) \wedge (\phi \vee \gamma) \in \Gamma_2}{\Gamma_2 \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\,\text{AXIOM}}{\Gamma_2, \psi \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\,\text{WEAKENING}}{\Gamma_2, \psi \vdash \phi \vee \gamma}\,\wedge\text{-E}_2 \qquad \mathcal{D}_2'' \qquad \mathcal{D}_2'''}{\Gamma_2, \psi \vdash \phi \vee (\psi \wedge \gamma)}\,\vee\text{-E}$$

We can derive $\mathcal{D}_2''$ as follows:

$$\dfrac{\dfrac{\phi \in \Gamma', \psi, \phi}{\Gamma', \psi, \phi \vdash \phi}\,\text{AXIOM}}{\Gamma', \psi, \phi \vdash \phi \vee (\psi \wedge \gamma)}\,\vee\text{-I}_2$$

And derivation $\mathcal{D}_2'''$ is done in the following way:

$$\dfrac{\dfrac{\dfrac{\dfrac{\psi \in \Gamma', \psi}{\Gamma', \psi \vdash \psi}\,\text{AXIOM}}{\Gamma', \psi, \gamma \vdash \psi}\,\text{WEAKENING} \qquad \dfrac{\gamma \in \Gamma', \psi, \gamma}{\Gamma', \psi, \gamma \vdash \gamma}\,\text{AXIOM}}{\Gamma', \psi, \gamma \vdash \psi \wedge \gamma}\,\wedge\text{-I}}{\Gamma', \psi, \gamma \vdash \phi \vee (\psi \wedge \gamma)}\,\vee\text{-I}_1$$

Thus we have also shown that $\Gamma, (\phi \vee \psi) \wedge (\phi \vee \gamma) \vdash \phi \vee (\psi \wedge \gamma)$. So the equivalence $\phi \vee (\psi \wedge \gamma) \sim (\phi \vee \psi) \wedge (\phi \vee \gamma)$ holds. □

Agda proof.

```
∨-dist1 : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ∨ (ψ ∧ γ) ⊢ (ϕ ∨ ψ) ∧ (ϕ ∨ γ)
∨-dist1 = ∨-E (axiom Z)
              (∧-I (∨-I₂ (axiom Z))
                   (∨-I₂ (axiom Z)))
              (∧-I (∨-I₁ (∧-E₁ (axiom Z)))
                   (∨-I₁ (∧-E₂ (axiom Z))))

∨-dist2 : ∀ {ϕ ψ γ : Formula} → Γ : (ϕ ∨ ψ) ∧ (ϕ ∨ γ) ⊢ ϕ ∨ (ψ ∧ γ)
∨-dist2 = ∨-E (∧-E₁ (axiom Z))
              (∨-I₂ (axiom Z))
              (∨-E (∧-E₂ (weakening (axiom Z)))
                   (∨-I₂ (axiom Z))
                   (∨-I₁ (∧-I (weakening (axiom Z)) (axiom Z))))

∼-dist-∨ : ∀ {ϕ ψ γ : Formula} → ϕ ∨ (ψ ∧ γ) ∼ (ϕ ∨ ψ) ∧ (ϕ ∨ γ)
∼-dist-∨ = ∨-dist1 , ∨-dist2
```

□

4 Lindenbaum-Tarski algebra in Cubical Agda

4.1 Definition

Definition 4.1 (Lindenbaum-Tarski algebra)**.** *The Lindenbaum-Tarski algebra is the quotient algebra obtained by factoring the algebra of formulas by the equivalence relation* $\sim$*.*

We use the definition of a set quotient already existing in the `agda/cubical` library to define the Lindenbaum-Tarski algebra.

Agda definition 17 (Lindenbaum-Tarski algebra)**.**

```
LindenbaumTarski : Type
LindenbaumTarski = Formula / _∼_
```

4.2 Binary operations and propositional constants

To define operations on the Lindenbaum-Tarski algebra, we utilize the already existing functions `setQuotBinOp` and `setQuotUnaryOp` from the `agda/cubical` library, which operate on set quotients. Our first step is to define conjunction in the Lindenbaum-Tarski algebra, using `setQuotBinOp` to create a binary operation. In order to do this, we must provide proof that the underlying operation - conjunction on formulas in the propositional calculus - respects the equivalence relation $\sim$. We will establish this as a lemma.

Lemma 4.1. *For all formulas* a, a', b, b'*, if* $a \sim a'$ *and* $b \sim b'$*, then* $a \wedge b \sim a' \wedge b'$*.*

Proof. By definition of the relation $\sim$ we have:

(i) $\Gamma, a \vdash a'$

(ii) $\Gamma, a' \vdash a$

(iii) $\Gamma, b \vdash b'$

(iv) $\Gamma, b' \vdash b$

We want to show that then $\Gamma, a \wedge b \vdash a' \wedge b'$ and $\Gamma, a' \wedge b' \vdash a \wedge b$. First, we show that $\Gamma, a \wedge b \vdash a' \wedge b'$. By the axiom rule and conjunction elimination, we have:

$$\dfrac{\dfrac{a \wedge b \in \Gamma, a \wedge b}{\Gamma, a \wedge b \vdash a \wedge b}\ \text{AXIOM}}{\Gamma, a \wedge b \vdash a}\ \wedge\text{-E}_1 \qquad \dfrac{\dfrac{a \wedge b \in \Gamma, a \wedge b}{\Gamma, a \wedge b \vdash a \wedge b}\ \text{AXIOM}}{\Gamma, a \wedge b \vdash b}\ \wedge\text{-E}_2$$

Now we can use Lemma 3.1 on $\Gamma, a \wedge b \vdash a$ and (i) to get $\Gamma, a \wedge b \vdash a'$, and similarly on $\Gamma, a \wedge b \vdash b$ and (iii) to get $\Gamma, a \wedge b \vdash b'$. Then, by conjunction introduction we get the desired result:

$$\dfrac{\Gamma, a \wedge b \vdash a' \qquad \Gamma, a \wedge b \vdash b'}{\Gamma, a \wedge b \vdash a' \wedge b'}\ \wedge\text{-I}$$

Proving $\Gamma, a' \wedge b' \vdash a \wedge b$ is identical to the first proof, so we omit it here. □

Agda proof.

```
∼-respects-∧ : ∀ (a a' b b' : Formula)
             → a ∼ a'
             → b ∼ b'
             → (a ∧ b) ∼ (a' ∧ b')
∼-respects-∧ a a' b b' x y =
             ∧-I (⊢trans (∧-E₁ (axiom Z)) (proj₁ x))
                 (⊢trans (∧-E₂ (axiom Z)) (proj₁ y)) ,
             ∧-I (⊢trans (∧-E₁ (axiom Z)) (proj₂ x))
                 (⊢trans (∧-E₂ (axiom Z)) (proj₂ y))
```

□

Now we are ready to define conjunction on Lindenbaum-Tarski algebra in `Cubical Agda`.

Agda definition 18 (Conjunction on Lindenbaum-Tarski algebra)**.**

```
_∧/_ : LindenbaumTarski → LindenbaumTarski → LindenbaumTarski
A ∧/ B = setQuotBinOp ∼-refl ∼-refl _∧_ ∼-respects-∧ A B
```

The `setQuotBinOp` function takes four arguments: two proofs of reflexivity, the logical conjunction operator $\wedge/$, and a proof that the operator respects the equivalence relation. In this case, the reflexivity proofs are identical since A and B are of the same type. The resulting function takes two elements A and B in the Lindenbaum-Tarski algebra and returns their conjunction.

When defining disjunction we need to prove a similar lemma as for the conjunction. We must provide proof that the disjunction operator respects the equivalence relation $\sim$.

Lemma 4.2. *For all formulas a, a', b, b', if $a \sim a'$ and $b \sim b'$, then $a \vee b \sim a' \vee b'$.*

Proof. By definition of the relation $\sim$, we have:

(i) $\Gamma, a \vdash a'$

(ii) $\Gamma, a' \vdash a$

(iii) $\Gamma, b \vdash b'$

(iv) $\Gamma, b' \vdash b$

We want to show that then $\Gamma, a \vee b \vdash a' \vee b'$ and $\Gamma, a' \vee b' \vdash a \vee b$. By the axiom rule and disjunction introduction, we have:

$$\frac{a \vee b \in \Gamma, a \vee b}{\Gamma, a \vee b \vdash a \vee b}\ \text{AXIOM}$$

Using (i) we can deduce:

$$\dfrac{\dfrac{\dfrac{\Gamma, a \vdash a'}{\Gamma, a \vdash a' \vee b'}\ \vee\text{-I}_2}{\Gamma, a, a \vee b \vdash a' \vee b'}\ \text{WEAKENING}}{\Gamma, a \vee b, a \vdash a' \vee b'}\ \text{EXCHANGE}$$

Similarly, using (iii) we can deduce:

$$\dfrac{\dfrac{\dfrac{\Gamma, b \vdash b'}{\Gamma, b \vdash a' \vee b'}\ \vee\text{-I}_2}{\Gamma, b, a \vee b \vdash a' \vee b'}\ \text{WEAKENING}}{\Gamma, a \vee b, b \vdash a' \vee b'}\ \text{EXCHANGE}$$

Then by using disjunction elimination we get the desired result,

$$\frac{\Gamma, a \vee b \vdash a \vee b \qquad \Gamma, a \vee b, a \vdash a' \vee b' \qquad \Gamma, a \vee b, b \vdash a' \vee b'}{\Gamma, a \vee b \vdash a' \vee b'}\ \vee\text{-E}$$

Proving $\Gamma, a' \vee b' \vdash a \vee b$ is identical to the first proof, so we omit it here. $\square$

Agda proof.

```
∼-respects-∨ : ∀ (a a' b b' : Formula)
             → a ∼ a'
             → b ∼ b'
             → (a ∨ b) ∼ (a' ∨ b')
∼-respects-∨ a a' b b' x y =
             ∨-E (axiom Z)
                 (exchange (weakening (∨-I₂ (proj₁ x))))
                 (exchange (weakening (∨-I₁ (proj₁ y)))) ,
             ∨-E (axiom Z)
                 (exchange (weakening (∨-I₂ (proj₂ x))))
                 (exchange (weakening (∨-I₁ (proj₂ y))))
```

□

Using `setQuotBinOp` we can now, with this lemma, define disjunction on Lindenbaum-Tarski algebra in `Cubical Agda`.

Agda definition 19 (Disjunction on Lindenbaum-Tarski algebra)**.**

```
_∨/_ : LindenbaumTarski → LindenbaumTarski → LindenbaumTarski
A ∨/ B = setQuotBinOp ∼-refl ∼-refl _∨_ ∼-respects-∨ A B
```

Next, in order to define negation on the Lindenbaum-Tarski algebra, we need to show that the underlying operation of negation in the propositional calculus respects the equivalence relation $\sim$. This can be achieved by proving another lemma.

Lemma 4.3. *For all formulas a, a', if $a \sim a'$, then $\neg a \sim \neg a'$.*

Proof. From the definition of $\sim$ we are given $\Gamma, a \vdash a'$ and $\Gamma, a' \vdash a$. We aim to show that $\Gamma, \neg a \vdash \neg a'$ and $\Gamma, \neg a' \vdash \neg a$. The two proofs are identical so we will only prove $\Gamma, \neg a \vdash \neg a'$ here. This is done through natural deduction,

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, a' \vdash a}{\Gamma, a', \neg a \vdash a}\,\text{WEAKENING}}{\Gamma, \neg a, a' \vdash a}\,\text{EXCHANGE} \qquad \dfrac{\dfrac{\neg a \in \Gamma, \neg a}{\Gamma, \neg a \vdash \neg a}\,\text{AXIOM}}{\Gamma, \neg a, a' \vdash \neg a}\,\text{WEAKENING}}{\Gamma, \neg a, a' \vdash \bot}\,\neg\text{-E}}{\Gamma, \neg a \vdash \neg a'}\,\neg\text{-I}$$

□

Agda proof.

```
∼-respects-¬ : ∀ (a a' : Formula)
             → a ∼ a'
```

```
                → (¬ a) ∼ (¬ a')
∼-respects-¬ a a' x =
                ¬-I (¬-E (exchange (weakening (proj₂ x)))
                         (weakening (axiom Z))) ,
                ¬-I (¬-E (exchange (weakening (proj₁ x)))
                         (weakening (axiom Z)))
```

□

This lemma shows that the negation operation on the propositional calculus respects the equivalence relation $\sim$, and allows us to define negation on the Lindenbaum-Tarski algebra in `Cubical Agda`.

Agda definition 20 (Negation on Lindenbaum-Tarski algebra)**.**

```
¬/_ : LindenbaumTarski → LindenbaumTarski
¬/ A = setQuotUnaryOp ¬_ ∼-respects-¬ A
```

The `setQuotUnaryOp` function takes two arguments: a unary operator ($\neg/$) and a proof that the operator respects the equivalence relation ($\sim$-respects-$\neg$). The resulting function takes one element A in the Lindenbaum-Tarski algebra and returns its negation.

To construct the equivalence classes for propositional constants, we can use the constructor from the definition of a set quotient in the already existing definition of set quotients in `Cubical Agda`.

Agda definition 21 (Logical constants in Lindenbaum-Tarski algebra)**.**

```
⊤/ : LindenbaumTarski
⊤/ = [ ⊤ ]

⊥/ : LindenbaumTarski
⊥/ = [ ⊥ ]
```

4.3 Proof that the Lindenbaum-Tarski algebra is Boolean

To prove that the Lindenbaum-Tarski algebra is Boolean we can show that it is a complemented distributive lattice. To do this there first needs to be a partial order defined on the set of equivalence classes.

Definition 4.2. *Let $\mathcal{L}$ be the propositional language described previously and let S be the set of all sentences of $\mathcal{L}$. For each $x \in S$, let $[x]$ denote the equivalence class of x under the equivalence relation $\sim$, and let LT be the set of all such equivalence classes. Define the relation $\leq$ on LT by*

$$[x] \leq [y] \quad \textit{iff} \quad \Gamma, x \vdash y$$

and define join, meet and complementation as follows:

$$[x] \vee [y] \equiv [x \vee y], \quad [x] \wedge [y] \equiv [x \wedge y], \quad \neg[x] \equiv [\neg x].$$

Note that the operators on the left hand side are operating on the equivalence classes, i.e. the elements of the Lindenbaum-Tarski algebra, and the operators on the right hand side are operating on formulas in the underlying propositional calculus.

Proposition 7. *The relation $\leq$ is a partial order on LT.*

Proof. First, we must show that the relation is well defined on LT. If $x_2 \in [x_1]$ and $y_2 \in [y_1]$, then

$$[x_1] \leq [y_1] \quad \text{implies} \quad [x_2] \leq [y_2],$$

or equivalently, using the definition above,

$$\Gamma, x_1 \vdash y_1 \quad \text{implies} \quad \Gamma, x_2 \vdash y_2$$

where Γ a context. Given $x_2 \in [x_1]$, we have that $x_1 \sim x_2$ which means that $\Gamma, x_1 \vdash x_2$ and $\Gamma, x_2 \vdash x_1$. Similarly, since $y_1 \sim y_2$, we have that $\Gamma, y_1 \vdash y_2$ and $\Gamma, y_2 \vdash y_1$. By applying Lemma 3.1 to $\Gamma, x_2 \vdash x_1$ and the assumption $\Gamma, x_1 \vdash y_1$, we get $\Gamma, x_2 \vdash y_1$. Applying the lemma again to the previous result together with $\Gamma, y_1 \vdash y_2$ yields $\Gamma, x_2 \vdash y_2$, showing that the relation is well defined. The operations join, meet and complementation have been shown to be well defined in the previous section when we defined the operations on the Lindenbaum-Tarski algebra.

Now, we need to show that $\leq$ is an order relation. Reflexivity follows immediately from the axiom rule. For anti-symmetry, suppose $x \leq y$ and $y \leq x$, then we have that $\Gamma, x \vdash y$ and $\Gamma, y \vdash x$, so by definition of the equivalence relation we have $x \sim y$. This means that $[x] \equiv [y]$, hence $\leq$ is anti-symmetric. Finally for transitivity assume that $x \leq y$ and $y \leq z$, then we have $\Gamma, x \vdash y$ and $\Gamma, y \vdash z$. Here we can apply Lemma 3.1 and get $\Gamma, x \vdash z$, that is $x \leq z$. Hence we can conclude that $\langle LT, \leq \rangle$ is a partially ordered set. □

Definition 4.3 (Lattice)**.** *A lattice is a non-empty partially ordered set $\langle L, \leq \rangle$ where every $x, y \in L$ has a supremum $x \vee y$, also called join, and an infimum $x \wedge y$, also called meet. It follows from this definition that*

$$x \leq y \quad \textit{iff} \quad x \vee y = y \quad \textit{iff} \quad x \wedge y = x.$$

Definition 4.4. *A lattice L is distributive if for all $x, y, z \in L$,*

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z),$$
$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z).$$

Definition 4.5. *A lattice L is complemented if there exist both least and greatest elements in L, denoted $\bot$ and $\top$, and for every $x \in L$ there exists $y \in L$ such that*

$$x \vee y = \top \quad \textit{and} \quad x \wedge y = \bot.$$

Definition 4.6. *A Boolean algebra is a complemented distributive lattice.*

Theorem 4.1. *The Lindenbaum-Tarski algebra is a Boolean algebra.*

Proof. First we must show that the $\langle LT, \leq \rangle$ is a lattice. To do this we must show that any two elements $[x], [y] \in LT$ has both a supremum and infimum, $[x] \vee [y]$ and $[x] \wedge [y]$ respectively.

To show that $[x] \vee [y]$ is an upper bound for $\{[x], [y]\}$ we need to show both that $[x] \leq [x \vee y]$ and that $[y] \leq [x \vee y]$. From reflexivity we have $[x] \leq [x]$, or $\Gamma, x \vdash x$. Using disjunction introduction we get $\Gamma, x \vdash x \vee y$, hence $[x] \leq [x \vee y]$ which is equivalent to $[x] \leq [x] \vee [y]$. With an identical argument we also get that $[y] \leq [x] \vee [y]$. To show that this is the least upper bound, assume that $[z]$ is any other upper bound for $\{[x], [y]\}$, then $[x] \leq [z]$ and $[y] \leq [z]$. This means that we have $\Gamma, x \vdash z$ and $\Gamma, y \vdash z$. From the axiom rule and disjunction elimination rule we can deduce

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Gamma, x \vdash z}}{\Gamma, x, x \vee y \vdash z}\,\text{WEAKENING}}{\Gamma, x \vee y, x \vdash z}\,\text{EXCHANGE} \qquad \dfrac{\dfrac{\dfrac{\mathcal{D}_2}{\Gamma, y \vdash z}}{\Gamma, y, x \vee y \vdash z}\,\text{WEAKENING}}{\Gamma, x \vee y, y \vdash z}\,\text{EXCHANGE}$$

$$\dfrac{\dfrac{}{\Gamma, x \vee y \vdash x \vee y}\,\text{AXIOM} \qquad \mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma, x \vee y \vdash z}\,\vee\text{-E}$$

Then $[x \vee y] \equiv [x] \vee [y] \leq [z]$. Hence $[x] \vee [y]$ is the least upper bound, i.e. the supremum.

Showing that $[x] \wedge [y]$ is the infimum is similar. It should be clear that from conjunction elimination we have both $\Gamma, x \wedge y \vdash x$ and $\Gamma, x \wedge y \vdash y$, so $[x \wedge y] \equiv [x] \wedge [y] \leq x$ and $[x \wedge y] \equiv [x] \wedge [y] \leq y$. Now assume that $[z]$ is any other lower bound for $\{[x], [y]\}$, then $[z] \leq [x]$ and $[z] \leq [y]$. This means that $\Gamma, z \vdash x$ and $\Gamma, z \vdash y$, and we can conclude that $\Gamma, z \vdash x \wedge y$, hence $[z] \leq [x \wedge y] \equiv [x] \wedge [y]$.

We have now shown that $\langle LT, \leq \rangle$ is a lattice. Distributivity follows from the distributive properties of propositional calculus, shown in Section 3.4.4. Having shown that $\langle LT, \leq \rangle$ is a distributive lattice, all that is left is to show that it is also complemented. In Definition 4.2 we defined complementation of an element $[x]$ to be $\neg[x] = [\neg x]$. This complement is unique[4], therefore we need to show that

$$[x] \vee \neg[x] \equiv [\top] \quad \text{and} \quad [x] \wedge \neg[x] \equiv [\bot].$$

Since we have that

$$[x] \vee \neg[x] \equiv [x] \vee [\neg x] \equiv [x \vee \neg x]$$

it suffices to show that $\Gamma, x \vee \neg x \vdash \top$ and $\Gamma, \top \vdash x \vee \neg x$. This is simply done with $\top$-I and LEM respectively. To show $[x] \wedge \neg[x] \equiv [\bot]$ we first we need to show $\Gamma, x \wedge \neg x \vdash \bot$, this is done by conjunction elimination and negation elimination,

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x \wedge \neg x \vdash x \wedge \neg x}\,\text{AXIOM}}{\Gamma, x \wedge \neg x \vdash x}\,\wedge\text{-E}_1 \qquad \dfrac{\dfrac{}{\Gamma, x \wedge \neg x \vdash x \wedge \neg x}\,\text{AXIOM}}{\Gamma, x \wedge \neg x \vdash \neg x}\,\wedge\text{-E}_2}{\Gamma, x \wedge \neg x \vdash \bot}\,\neg\text{-E}$$

The other direction, i.e $\Gamma, \bot \vdash x \wedge \neg x$, is just applying $\bot$-E. This shows that $\langle LT, \leq \rangle$ is a complemented distributive lattice, and thus a Boolean algebra. $\square$

Agda proof. We make use of the fact that there is already a function for constructing a distributive lattice in the `agda/cubical` library. We need only provide all the proofs needed for the arguments.

```
LindenbaumTarski-DistLattice : DistLattice _
LindenbaumTarski-DistLattice = makeDistLattice∧lOver∨l
                                 ⊥/
                                 ⊤/
                                 _∨/_
                                 _∧/_
                                 isSet-LT
                                 ∨/-ass
                                 ∨/-id
                                 ∨/-comm
                                 ∧/-ass
                                 ∧/-id
                                 ∧/-comm
                                 ∧/-abs
                                 ∧/-dist
  where
    isSet-LT : ∀ (A B : LindenbaumTarski) → isProp(A ≡ B)
    isSet-LT A B = squash/ _ _

    ∧/-comm : ∀ (A B : LindenbaumTarski) → A ∧/ B ≡ B ∧/ A
    ∧/-comm = elimProp2 (λ _ _ → squash/ _ _)
                        λ _ _ → eq/ _ _ ∼-comm-∧

    ∨/-comm : ∀ (A B : LindenbaumTarski) → A ∨/ B ≡ B ∨/ A
    ∨/-comm = elimProp2 (λ _ _ → squash/ _ _)
                        λ _ _ → eq/ _ _ ∼-comm-∨

    ∧/-ass : ∀ (A B C : LindenbaumTarski)
           → A ∧/ (B ∧/ C) ≡ (A ∧/ B) ∧/ C
    ∧/-ass = elimProp3 (λ _ _ _ → squash/ _ _)
                       λ _ _ _ → eq/ _ _ ∼-ass-∧

    ∨/-ass : ∀ (A B C : LindenbaumTarski)
           → A ∨/ (B ∨/ C) ≡ (A ∨/ B) ∨/ C
    ∨/-ass = elimProp3 (λ _ _ _ → squash/ _ _)
                       λ _ _ _ → eq/ _ _ ∼-ass-∨

    ∧/-dist : ∀ (A B C : LindenbaumTarski)
            → A ∧/ (B ∨/ C) ≡ (A ∧/ B) ∨/ (A ∧/ C)
    ∧/-dist = elimProp3 (λ _ _ _ → squash/ _ _)
```

```
                                λ _ _ _ → eq/ _ _ ∼-dist-∧

  ∨/-dist : ∀ (A B C : LindenbaumTarski)
          → A ∨/ (B ∧/ C) ≡ (A ∨/ B) ∧/ (A ∨/ C)
  ∨/-dist = elimProp3 (λ _ _ _ → squash/ _ _)
                      λ _ _ _ → eq/ _ _ ∼-dist-∨

  ∧/-abs : ∀ (A B : LindenbaumTarski) → A ∧/ (A ∨/ B) ≡ A
  ∧/-abs = elimProp2 (λ _ _ → squash/ _ _)
                     λ _ _ → eq/ _ _
                     (∧-E₁ (axiom Z) , ∧-I (axiom Z) (∨-I₂ (axiom Z)))

  ∨/-abs : ∀ (A B : LindenbaumTarski) → (A ∧/ B) ∨/ B ≡ B
  ∨/-abs = elimProp2 (λ _ _ → squash/ _ _)
                     λ _ _ → eq/ _ _
                     (∨-E (axiom Z) (∧-E₂ (axiom Z)) (axiom Z) ,
                     ∨-I₁ (axiom Z))

  ∨/-id : ∀ (A : LindenbaumTarski) → A ∨/ ⊥/ ≡ A
  ∨/-id = elimProp (λ _ → squash/ _ _)
                   λ _ → eq/ _ _
                   (∨-E (axiom Z) (axiom Z) (⊥-E (axiom Z)) ,
                   ∨-I₂ (axiom Z))

  ∧/-id : ∀ (A : LindenbaumTarski) → A ∧/ ⊤/ ≡ A
  ∧/-id = elimProp (λ _ → squash/ _ _)
                   λ _ → eq/ _ _
                   (∧-E₁ (axiom Z) , ∧-I (axiom Z) ⊤-I)
```

Lastly we also need to provide the proof that the lattice is complemented.

```
  open DistLatticeStr (snd LindenbaumTarski-DistLattice)

  LindenbaumTarski-DistLattice-supremum :
    (x : fst LindenbaumTarski-DistLattice) → x ∨l ¬/ x ≡ 1l
  LindenbaumTarski-DistLattice-supremum x = ∨/-comp x
    where
      ∨/-comp : ∀ (A : LindenbaumTarski) → A ∨/ ¬/ A ≡ ⊤/
      ∨/-comp = elimProp (λ _ → squash/ _ _) λ _ → eq/ _ _ (⊤-I , LEM)

  LindenbaumTarski-DistLattice-infimum :
    (x : fst LindenbaumTarski-DistLattice) → x ∧l ¬/ x ≡ 0l
  LindenbaumTarski-DistLattice-infimum x = ∧/-comp x
    where
      ∧/-comp : ∀ (A : LindenbaumTarski) → A ∧/ ¬/ A ≡ ⊥/
      ∧/-comp = elimProp (λ _ → squash/ _ _)
                  λ _ → eq/ _ _ (¬-E (∧-E₁ (axiom Z)) (∧-E₂ (axiom Z)) ,
                                 ⊥-E (axiom Z))
```

□

4.4 Soundness

Soundness is a fundamental property of logical systems that ensures that the conclusions drawn from the system are reliable and trustworthy. In particular, soundness guarantees that if a proposition can be derived using the inference rules of the logical system from a set of true premises, then that proposition is true.

Theorem 4.2. *The Lindenbaum-Tarski algebra is sound.*

Proof. To prove that the Lindenbaum-Tarski algebra is sound, we need to show that any formula that is provable also belongs to the same equivalence class as $\top$. In other words, we need to show that if $\Gamma \vdash \phi$ then $[\phi] \equiv [\top]$. By definition of the Lindenbaum-Tarski algebra this is equivalent to showing $\Gamma, \phi \vdash \top$ and $\Gamma, \top \vdash \phi$. This can be done using $\top$-I and weakening respectively.

$$\frac{}{\Gamma, \phi \vdash \top}\ \top\text{-I} \qquad \frac{\Gamma \vdash \phi}{\Gamma, \top \vdash \phi}\ \text{WEAKENING}$$

□

Agda proof.

```
sound : ∀ {ϕ : Formula} → Γ ⊢ ϕ → [ ϕ ] ≡ ⊤/
sound x = eq/ _ _ (⊤-I , weakening x)
```

□

5 Conclusions

We successfully formalized the Lindenbaum-Tarski algebra in `Cubical Agda`

5.1 Possible improvements

We defined the notion of a context as a set of formulas and implemented it as a list in `Agda`. To improve the implementation of the structural rule exchange, it is worth exploring the possibility of exchanging any two elements in the context, rather than just the last two. To do this one might need to choose another way to represent the context, perhaps as a vector.

Additionally, we could consider modifying the deduction rules for the propositional calculus, such as adding implication, to see if this reduces the dependency on the structural rules.

5.2 Future work

We have formalized the Lindenbaum-Tarski algebra over a propositional language, the next step would be to define it over a first order language. The equivalence relation $\sim$ and the binary relations together with negation is defined as previously, but we would need to account for the quantifiers.

References

[1] Agda documentation, 2023. URL: https://agda.readthedocs.io/en/v2.6.3/index.html.

[2] Ana Bove and Peter Dybjer. Dependent types at work, 2008. URL: https://www.cse.chalmers.se/~peterd/papers/DependentTypesAtWork.pdf.

[3] Jesper Carlström. *Logic*. 2008.

[4] Sam Chong Tay. Logic via algebra: A senior exercise in mathematics, 2012.

[5] Dick van Dalen. *Logic and structure*. Springer, fifth edition, 2013.

[6] Wikipedia. Lindenbaum-tarski algebra. Retrieved 2023-04-28. URL: https://en.wikipedia.org/wiki/Lindenbaum%E2%80%93Tarski_algebra.

A LindenbaumTarski.agda

```
{-# OPTIONS --cubical --safe #-}
module LindenbaumTarski where

open import Cubical.HITs.SetQuotients.Base
open import Cubical.HITs.SetQuotients.Properties
open import Cubical.Foundations.Prelude hiding (_∧_; _∨_)
open import Cubical.Relation.Binary.Base
open import Cubical.Data.Nat.Base
open import Cubical.Data.Prod.Base
open import Cubical.Algebra.DistLattice.Base


-- Definition: Formula


data Formula : Type where
  _∧_ : Formula → Formula → Formula
  _∨_ : Formula → Formula → Formula
  ¬_  : Formula → Formula
  const : ℕ → Formula
  ⊥   : Formula
  ⊤   : Formula


infix 35 _∧_
infix 30 _∨_
infixl 36 ¬_
infix 20 _⊢_
infix 23 _:_


-- Definition: Context


data ctxt : Type where
  ∅   : ctxt
  _:_ : ctxt → Formula → ctxt


-- Definition: Lookup


data _∈_ : Formula → ctxt → Type where
  Z : ∀ {Γ ϕ} → ϕ ∈ Γ : ϕ
  S : ∀ {Γ ϕ ψ} → ϕ ∈ Γ → ϕ ∈ Γ : ψ


-- Definition: Provability
```

```
data ⊢_ : ctxt → Formula → Type where

∧-I : {Γ : ctxt} {ϕ ψ : Formula}
    → Γ ⊢ ϕ
    → Γ ⊢ ψ
    → Γ ⊢ ϕ ∧ ψ

∧-E₁ : {Γ : ctxt} {ϕ ψ : Formula}
     → Γ ⊢ ϕ ∧ ψ
     → Γ ⊢ ϕ

∧-E₂ : {Γ : ctxt} {ϕ ψ : Formula}
     → Γ ⊢ ϕ ∧ ψ
     → Γ ⊢ ψ

∨-I₁ : {Γ : ctxt} {ϕ ψ : Formula}
     → Γ ⊢ ψ
     → Γ ⊢ ϕ ∨ ψ

∨-I₂ : {Γ : ctxt} {ϕ ψ : Formula}
     → Γ ⊢ ϕ
     → Γ ⊢ ϕ ∨ ψ

∨-E : {Γ : ctxt} {ϕ ψ γ : Formula}
     → Γ ⊢ ϕ ∨ ψ
     → Γ : ϕ ⊢ γ
     → Γ : ψ ⊢ γ
     → Γ ⊢ γ

¬-I : {Γ : ctxt} {ϕ : Formula}
    → Γ : ϕ ⊢ ⊥
    → Γ ⊢ ¬ ϕ

¬-E : {Γ : ctxt} {ϕ : Formula}
     → Γ ⊢ ϕ
     → Γ ⊢ ¬ ϕ
     → Γ ⊢ ⊥

⊥-E : {Γ : ctxt} {ϕ : Formula}
     → Γ ⊢ ⊥
     → Γ ⊢ ϕ
```

```
⊤-I : {Γ : ctxt} → Γ ⊢ ⊤


axiom : {Γ : ctxt} {ϕ : Formula}
      → ϕ ∈ Γ
      → Γ ⊢ ϕ


LEM : {Γ : ctxt} {ϕ : Formula}
    → Γ ⊢ ϕ ∨ ¬ ϕ


weakening : {Γ : ctxt} {ϕ ψ : Formula}
          → Γ ⊢ ψ
          → Γ : ϕ ⊢ ψ


exchange : {Γ : ctxt} {ϕ ψ γ : Formula}
         → (Γ : ϕ) : ψ ⊢ γ
         → (Γ : ψ) : ϕ ⊢ γ


--  contraction : {Γ : ctxt} {ϕ ψ : Formula}
--              → ((Γ : ϕ) : ϕ) ⊢ ψ
--              → (Γ : ϕ) ⊢ ψ


module _ {Γ : ctxt} where

  infixl 25 ¬/_


  ------------------------------------------------------
  -- Defining relation where two formulas are related
  -- if they are provably equivalent. Then proving that
  -- the relation is an equivalence relation by proving
  -- it is reflexive, symmetric and transitive.
  ------------------------------------------------------


_∼_ : Formula → Formula → Type
ϕ ∼ ψ = Γ : ϕ ⊢ ψ × Γ : ψ ⊢ ϕ


∼-refl : ∀ (ϕ : Formula) → ϕ ∼ ϕ
∼-refl _ = axiom Z , (axiom Z)


∼-sym : ∀ {ϕ ψ : Formula} → ϕ ∼ ψ → ψ ∼ ϕ
∼-sym (A , B) = B , A
```

```
⊢trans : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ⊢ γ → Γ : γ ⊢ ψ → Γ : ϕ ⊢ ψ
⊢trans A B = ∨-E (∨-I₂ A) (exchange (weakening B)) (axiom Z)


∼-trans : ∀ {ϕ ψ γ : Formula} → ϕ ∼ γ → γ ∼ ψ → ϕ ∼ ψ
∼-trans x y = ⊢trans (proj₁ x) (proj₁ y) , ⊢trans (proj₂ y) (proj₂ x)


----------------------------------------
-- Properties of propositional calculus
----------------------------------------


-- Commutativity on ∧


∧-comm : ∀ {ϕ ψ : Formula} → Γ : ϕ ∧ ψ ⊢ ψ ∧ ϕ
∧-comm = ∧-I (∧-E₂ (axiom Z)) (∧-E₁ (axiom Z))


∼-comm-∧ : ∀ {ϕ ψ : Formula} → ϕ ∧ ψ ∼ ψ ∧ ϕ
∼-comm-∧ = ∧-comm , ∧-comm


-- Commutativity on ∨


∨-comm : {ϕ ψ : Formula} → Γ : ϕ ∨ ψ ⊢ ψ ∨ ϕ
∨-comm = ∨-E (axiom Z) (∨-I₁ (axiom Z)) (∨-I₂ (axiom Z))


∼-comm-∨ : ∀ {ϕ ψ : Formula} → ϕ ∨ ψ ∼ ψ ∨ ϕ
∼-comm-∨ = ∨-comm , ∨-comm


-- Associativity on ∧


∧-ass1 : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ∧ (ψ ∧ γ) ⊢ (ϕ ∧ ψ) ∧ γ
∧-ass1 = ∧-I (∧-I (∧-E₁ (axiom Z)) (∧-E₁ (∧-E₂ (axiom Z))))
             (∧-E₂ (∧-E₂ (axiom Z)))


∧-ass2 : ∀ {ϕ ψ γ : Formula} → Γ : (ϕ ∧ ψ) ∧ γ ⊢ ϕ ∧ (ψ ∧ γ)
∧-ass2 = ∧-I (∧-E₁ (∧-E₁ (axiom Z)))
             (∧-I (∧-E₂ (∧-E₁ (axiom Z)))
                  (∧-E₂ (axiom Z)))


∼-ass-∧ : ∀ {ϕ ψ γ : Formula} → ϕ ∧ (ψ ∧ γ) ∼ (ϕ ∧ ψ) ∧ γ
∼-ass-∧ = ∧-ass1 , ∧-ass2
```

```
-- Associativity on ∨

∨-ass1 : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ∨ (ψ ∨ γ) ⊢ (ϕ ∨ ψ) ∨ γ
∨-ass1 = ∨-E (axiom Z)
             (∨-I₂ (∨-I₂ (axiom Z)))
             (∨-E (axiom Z)
                  (∨-I₂ (∨-I₁ (axiom Z)))
                  (∨-I₁ (axiom Z)))

∨-ass2 : ∀ {ϕ ψ γ : Formula} → Γ : (ϕ ∨ ψ) ∨ γ ⊢ ϕ ∨ (ψ ∨ γ)
∨-ass2 = ∨-E (axiom Z)
             (∨-E (axiom Z)
                  (∨-I₂ (axiom Z))
                  (∨-I₁ (∨-I₂ (axiom Z))))
             (∨-I₁ (∨-I₁ (axiom Z)))

∼-ass-∨ : ∀ {ϕ ψ γ : Formula} → ϕ ∨ (ψ ∨ γ) ∼ (ϕ ∨ ψ) ∨ γ
∼-ass-∨ = ∨-ass1 , ∨-ass2

-- Distributivity over ∧

∧-dist1 : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ∧ (ψ ∨ γ) ⊢ (ϕ ∧ ψ) ∨ (ϕ ∧ γ)
∧-dist1 = ∨-E (∧-E₂ (axiom Z))
              (∨-I₂ (∧-I (weakening (∧-E₁ (axiom Z))) (axiom Z)))
              (∨-I₁ (∧-I (weakening (∧-E₁ (axiom Z))) (axiom Z)))

∧-dist2 : ∀ {ϕ ψ γ : Formula} → Γ : (ϕ ∧ ψ) ∨ (ϕ ∧ γ) ⊢ ϕ ∧ (ψ ∨ γ)
∧-dist2 = ∧-I (∨-E (axiom Z)
                   (∧-E₁ (axiom Z))
                   (∧-E₁ (axiom Z)))
              (∨-E (axiom Z)
                   (∨-I₂ (∧-E₂ (axiom Z)))
                   (∨-I₁ (∧-E₂ (axiom Z))))

∼-dist-∧ : ∀ {ϕ ψ γ : Formula} → ϕ ∧ (ψ ∨ γ) ∼ (ϕ ∧ ψ) ∨ (ϕ ∧ γ)
∼-dist-∧ = ∧-dist1 , ∧-dist2

-- Distributivity over ∨

∨-dist1 : ∀ {ϕ ψ γ : Formula} → Γ : ϕ ∨ (ψ ∧ γ) ⊢ (ϕ ∨ ψ) ∧ (ϕ ∨ γ)
∨-dist1 = ∨-E (axiom Z)
```

```
                (∧-I (∨-I₂ (axiom Z))
                     (∨-I₂ (axiom Z)))
                (∧-I (∨-I₁ (∧-E₁ (axiom Z)))
                     (∨-I₁ (∧-E₂ (axiom Z))))


∨-dist2 : ∀ {ϕ ψ γ : Formula} → Γ : (ϕ ∨ ψ) ∧ (ϕ ∨ γ) ⊢ ϕ ∨ (ψ ∧ γ)
∨-dist2 = ∨-E (∧-E₁ (axiom Z))
              (∨-I₂ (axiom Z))
              (∨-E (∧-E₂ (weakening (axiom Z)))
                   (∨-I₂ (axiom Z))
                   (∨-I₁ (∧-I (weakening (axiom Z)) (axiom Z))))


∼-dist-∨ : ∀ {ϕ ψ γ : Formula} → ϕ ∨ (ψ ∧ γ) ∼ (ϕ ∨ ψ) ∧ (ϕ ∨ γ)
∼-dist-∨ = ∨-dist1 , ∨-dist2


-------------------------------------------------------
-- Lindenbaum-Tarski algebra is defined as the quotioent
-- algebra obtained by factoring the algebra of formulas
-- by the above defined equivalence relation.
-------------------------------------------------------


LindenbaumTarski : Type
LindenbaumTarski = Formula / _∼_


-------------------------------------------------
-- The equivalence relation ∼ respects operations
-------------------------------------------------


∼-respects-∧ : ∀ (a a' b b' : Formula)
             → a ∼ a'
             → b ∼ b'
             → (a ∧ b) ∼ (a' ∧ b')
∼-respects-∧ a a' b b' x y =
             ∧-I (⊢trans (∧-E₁ (axiom Z)) (proj₁ x))
                 (⊢trans (∧-E₂ (axiom Z)) (proj₁ y)) ,
             ∧-I (⊢trans (∧-E₁ (axiom Z)) (proj₂ x))
                 (⊢trans (∧-E₂ (axiom Z)) (proj₂ y))


∼-respects-∨ : ∀ (a a' b b' : Formula)
             → a ∼ a'
             → b ∼ b'
             → (a ∨ b) ∼ (a' ∨ b')
```

```
∼-respects-∨ a a' b b' x y =
             ∨-E (axiom Z)
                 (exchange (weakening (∨-I₂ (proj₁ x))))
                 (exchange (weakening (∨-I₁ (proj₁ y)))) ,
             ∨-E (axiom Z)
                 (exchange (weakening (∨-I₂ (proj₂ x))))
                 (exchange (weakening (∨-I₁ (proj₂ y))))


∼-respects-¬ : ∀ (a a' : Formula)
             → a ∼ a'
             → (¬ a) ∼ (¬ a')
∼-respects-¬ a a' x =
             ¬-I (¬-E (exchange (weakening (proj₂ x)))
                      (weakening (axiom Z))) ,
             ¬-I (¬-E (exchange (weakening (proj₁ x)))
                      (weakening (axiom Z)))


-----------------------------------------------------------------------
-- Definition: Binary operations and propositional constants in LT
-----------------------------------------------------------------------


_∧/_ : LindenbaumTarski → LindenbaumTarski → LindenbaumTarski
A ∧/ B = setQuotBinOp ∼-refl ∼-refl _∧_ ∼-respects-∧ A B


_∨/_ : LindenbaumTarski → LindenbaumTarski → LindenbaumTarski
A ∨/ B = setQuotBinOp ∼-refl ∼-refl _∨_ ∼-respects-∨ A B


¬/_ : LindenbaumTarski → LindenbaumTarski
¬/ A = setQuotUnaryOp ¬_ ∼-respects-¬ A


⊤/ : LindenbaumTarski
⊤/ = [ ⊤ ]


⊥/ : LindenbaumTarski
⊥/ = [ ⊥ ]


-----------------------------------------------------------
-- If φ is provable in Γ then [φ] should be the same as ⊤/.
-- We can view this as a form of soundness.
-----------------------------------------------------------
```

```
sound : ∀ {ϕ : Formula} → Γ ⊢ ϕ → [ ϕ ] ≡ ⊤/
sound x = eq/ _ _ (⊤-I , weakening x)


------------------------------------------------------------
-- By proving the Lindenbaum-Tarski algebra can be viewed as
-- a complemented distributive lattice, we prove that it is
-- also boolean.
------------------------------------------------------------


LindenbaumTarski-DistLattice : DistLattice _
LindenbaumTarski-DistLattice = makeDistLattice∧lOver∨l
                                 ⊥/
                                 ⊤/
                                 _∨/_
                                 _∧/_
                                 isSet-LT
                                 ∨/-ass
                                 ∨/-id
                                 ∨/-comm
                                 ∧/-ass
                                 ∧/-id
                                 ∧/-comm
                                 ∧/-abs
                                 ∧/-dist
  where
    isSet-LT : ∀ (A B : LindenbaumTarski) → isProp(A ≡ B)
    isSet-LT A B = squash/ _ _

    ∧/-comm : ∀ (A B : LindenbaumTarski) → A ∧/ B ≡ B ∧/ A
    ∧/-comm = elimProp2 (λ _ _ → squash/ _ _)
                        λ _ _ → eq/ _ _ ∼-comm-∧

    ∨/-comm : ∀ (A B : LindenbaumTarski) → A ∨/ B ≡ B ∨/ A
    ∨/-comm = elimProp2 (λ _ _ → squash/ _ _)
                        λ _ _ → eq/ _ _ ∼-comm-∨

    ∧/-ass : ∀ (A B C : LindenbaumTarski)
           → A ∧/ (B ∧/ C) ≡ (A ∧/ B) ∧/ C
    ∧/-ass = elimProp3 (λ _ _ _ → squash/ _ _)
                       λ _ _ _ → eq/ _ _ ∼-ass-∧

    ∨/-ass : ∀ (A B C : LindenbaumTarski)
           → A ∨/ (B ∨/ C) ≡ (A ∨/ B) ∨/ C
    ∨/-ass = elimProp3 (λ _ _ _ → squash/ _ _)
                       λ _ _ _ → eq/ _ _ ∼-ass-∨
```

```
    ∧/-dist : ∀ (A B C : LindenbaumTarski)
            → A ∧/ (B ∨/ C) ≡ (A ∧/ B) ∨/ (A ∧/ C)
    ∧/-dist = elimProp3 (λ _ _ _ → squash/ _ _)
                        λ _ _ _ → eq/ _ _ ∼-dist-∧

    ∨/-dist : ∀ (A B C : LindenbaumTarski)
            → A ∨/ (B ∧/ C) ≡ (A ∨/ B) ∧/ (A ∨/ C)
    ∨/-dist = elimProp3 (λ _ _ _ → squash/ _ _)
                        λ _ _ _ → eq/ _ _ ∼-dist-∨

    ∧/-abs : ∀ (A B : LindenbaumTarski) → A ∧/ (A ∨/ B) ≡ A
    ∧/-abs = elimProp2 (λ _ _ → squash/ _ _)
                       λ _ _ → eq/ _ _
                       (∧-E₁ (axiom Z) , ∧-I (axiom Z) (∨-I₂ (axiom Z)))

    ∨/-abs : ∀ (A B : LindenbaumTarski) → (A ∧/ B) ∨/ B ≡ B
    ∨/-abs = elimProp2 (λ _ _ → squash/ _ _)
                       λ _ _ → eq/ _ _
                       (∨-E (axiom Z) (∧-E₂ (axiom Z)) (axiom Z) ,
                       ∨-I₁ (axiom Z))

    ∨/-id : ∀ (A : LindenbaumTarski) → A ∨/ ⊥/ ≡ A
    ∨/-id = elimProp (λ _ → squash/ _ _)
                     λ _ → eq/ _ _
                     (∨-E (axiom Z) (axiom Z) (⊥-E (axiom Z)) ,
                     ∨-I₂ (axiom Z))

    ∧/-id : ∀ (A : LindenbaumTarski) → A ∧/ ⊤/ ≡ A
    ∧/-id = elimProp (λ _ → squash/ _ _)
                     λ _ → eq/ _ _
                     (∧-E₁ (axiom Z) , ∧-I (axiom Z) ⊤-I)


open DistLatticeStr (snd LindenbaumTarski-DistLattice)

LindenbaumTarski-DistLattice-supremum :
  (x : fst LindenbaumTarski-DistLattice) → x ∨l ¬/ x ≡ 1l
LindenbaumTarski-DistLattice-supremum x = ∨/-comp x
  where
    ∨/-comp : ∀ (A : LindenbaumTarski) → A ∨/ ¬/ A ≡ ⊤/
    ∨/-comp = elimProp (λ _ → squash/ _ _) λ _ → eq/ _ _ (⊤-I , LEM)

LindenbaumTarski-DistLattice-infimum :
  (x : fst LindenbaumTarski-DistLattice) → x ∧l ¬/ x ≡ 0l
LindenbaumTarski-DistLattice-infimum x = ∧/-comp x
  where
    ∧/-comp : ∀ (A : LindenbaumTarski) → A ∧/ ¬/ A ≡ ⊥/
    ∧/-comp = elimProp (λ _ → squash/ _ _)
```

$\lambda \_ \to \mathsf{eq}/\ \_\ \_\ (\neg\text{-E}\ (\wedge\text{-E}_1\ (\mathsf{axiom}\ Z))\ (\wedge\text{-E}_2\ (\mathsf{axiom}\ Z))\ ,$
$\bot\text{-E}\ (\mathsf{axiom}\ Z))$