Abstract

Contents

1 Introduction

2 Agda proof assistant

Agda is a dependently typed programming language based on intuitionistic type theory. By encoding mathematical propositions as types and their proofs as programs, we can ensure that our reasoning is correct and consistent. Agda's type system also provides powerful tools for automatically checking the correctness of proofs[2].

2.1 Propositions as types

Propositions as types associates 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 seen as operations that construct and deconstruct values of the corresponding types. For example, the introduction rule for conjunction says that if we have 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.

On the other hand, the elimination rule for conjunction says that if we have a proof of a conjunction, we can extract proofs of its two conjuncts by projecting the pair onto each component. This can be seen 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 is a type that 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. This chart summarizes the correspondence between proposition 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 datatypes

A data declaration is used to introduce datatypes, including their name, type, and constructors along with their types. An example of this is the declaration of the boolean type:

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

This states that `Bool` is a data type with `true` and `false` as constructors. Functions over this datatype `Bool` can be defined using pattern matching, similar to Haskell. For instance we can define a function `not` for `Bool` 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 using pattern matching on the arguments. Agda checks that the pattern covers all cases and will not accept a function with missing patterns.

The natural numbers can be defined as the datatype:

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

A natural number is either zero or a successor of another natural number. This is called an *inductively defined type*. We can define addition on the natural numbers with a recursive function.

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

If a name contains underscores (_) in the definition, the underscores 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 _+_
```

Datatypes 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
```

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) → 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} → B means the same thing as (A : Set) → 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 dealing with quotient types, and for that we need to use `Cubical Agda`. This extends `Agda` with features from Cubical Type Theory, which is needed when we deal with set quotients.[1] We use only a small part of the `agda/cubical` library, and any cubical theory is beyond 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 of propositional calculus 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 semantics of propositional calculus define how the expressions in the system correspond to truth values, typically "true" or "false".

3.1 Formulas

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 $\to$ and $\leftrightarrow$. This is becuase we can define them using other connectives,

$$\phi \to \psi \overset{\mathsf{def}}{=} \neg\phi \vee \psi,$$
$$\phi \leftrightarrow \psi \overset{\mathsf{def}}{=} (\neg\phi \vee \psi) \wedge (\neg\psi \vee \phi),$$

making them reduntant. It is possible to choose an even smaller set of connectives [3], but we choose this 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*

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

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.*

We define well formed formulas in Agda as a data type:

```
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 will interpret $\Gamma \cup \{\phi\}$ as $\Gamma : \phi$. Lets define the data type for context in `Agda`:

```
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 (Lookup)**.** *For all contexts Γ and all formulas ϕ and ψ*

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

We represent this as a data type in `Agda` as follows:

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

3.3 Inference rules

For the inference rules we introduce a data type for provability:

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

                    ⋮
```

We will define the inference rules as inhabitants of this type. They will be on the form:

```
rulename : { ... : ctxt} { ... : Formula}
        -> premise.1
        -> premise.2
              :
        -> premise.n
        -> conclusion
```

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.*

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

In `Agda`, we define this as follows:

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

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. These rules are based on [3].

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.

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

The conjunction introduction rule is formalised in `Agda` as follows:

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

The accompanying 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.

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

In `Agda`, the conjunction eliminations rules are defined as follows:

```
∧-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 introductory rules. If we can deduce som fomula ψ from Γ, then we can also deduce $\phi \vee \psi$ from Γ. In the same way, if we can deduce ψ from Γ, then we can decuce $\phi \vee \psi$ from Γ as well.

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

We formalise the two disjunction introduction rules in `Agda` as follows:

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


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

The eliminination 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.

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

The disjunction elimination rule in `Agda`:

```
∨-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.

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

In `Agda`, we define the negation introduction rule as follows:

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

The negation elimination rule is defined in `Agda` as follows:

```
¬-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 trivially true in all contexts, and should be introduced with no premise.

$$\frac{}{\Gamma \vdash \top}\ \top\ \mathrm{I}$$

The $\top$ introduction rule in `Agda`:

```
⊤-I : {Γ : ctxt}
    → Γ ⊢ ⊤
```

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

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

The $\bot$ elimination rule is formalised in `Agda` as follows:

```
⊥-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,

$$\frac{\Gamma \vdash \phi}{\Gamma, \psi \vdash \phi}\ \textsc{weakening}$$

In `Agda`, we define the rule of weakening as follows:

```
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.

$$\frac{\Gamma, \phi, \psi \vdash \gamma}{\Gamma, \psi, \phi \vdash \gamma}\ \textsc{exchange}$$

We define this exchange rule in `Agda` as follows:

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

3.4 Properties of a propositional calculus

We prove some properties of propositional calculus

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 $\phi, \psi \in S$,*

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

This is represented in `Agda` as a product type.

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

Before we prove that this is an equivalence relation we will prove a 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

$$\dfrac{\dfrac{\Gamma, \phi \vdash \psi}{\Gamma, \phi \vdash \psi \vee \gamma}\ \vee\text{-I}_2 \qquad \dfrac{\dfrac{\Gamma, \psi \vdash \gamma}{\Gamma, \psi, \phi \vdash \gamma}\ \textsc{weakening}}{\Gamma, \phi, \psi \vdash \gamma}\ \textsc{exchange} \qquad \dfrac{\gamma \in \Gamma, \phi, \gamma}{\Gamma, \phi, \gamma \vdash \gamma}\ \textsc{axiom}}{\Gamma, \phi \vdash \gamma}\ \vee\text{-E}$$

The corresponding `Agda` proof of the lemma:

```
⊢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. Reflexivity follows from immediately from the axiom rule:

$$\frac{\phi \in \Gamma, \phi}{\Gamma, \phi \vdash \phi}\ \textsc{axiom}$$

The corresponding `Agda` proof for reflexivity:

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

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. We formalise this in `Agda` as follows:

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

For transitivity we need to prove 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) $\Gamma, \gamma \vdash \psi$, and

(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 $\Gamma, \psi \vdash \phi$. The proof in `Agda`:

```
∼-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. $\phi \wedge \psi \sim \psi \wedge \phi$

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 have:

$$\cfrac{\cfrac{\cfrac{\phi \wedge \psi \in \Gamma, \phi \wedge \psi}{\Gamma, \phi \wedge \psi \vdash \phi \wedge \psi}\ \textsc{axiom}}{\Gamma, \phi \wedge \psi \vdash \psi}\ \wedge\text{-E}_2 \qquad \cfrac{\cfrac{\phi \wedge \psi \in \Gamma, \phi \wedge \psi}{\Gamma, \phi \wedge \psi \vdash \phi \wedge \psi}\ \textsc{axiom}}{\Gamma, \phi \wedge \psi \vdash \phi}\ \wedge\text{-E}_1}{\Gamma \vdash \psi \wedge \phi}\ \wedge\text{-I}$$

The other proof is identical up to renaming of the formulas, so we omitt it. Together they prove $\phi \wedge \psi \sim \psi \wedge \phi$. This corresponds to the `Agda` proof:

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

Thus, we have shown that the two statements are equivalent in `Agda`:

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

□

Proposition 2. $\phi \vee \psi \sim \psi \vee \phi$

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 can show $\Gamma' \vdash \psi \vee \phi$.

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

Therefore, we have shown $\Gamma, \phi \vee \psi \vdash \psi \vee \phi$ and $\Gamma, \psi \vee \phi \vdash \phi \vee \psi$. Hence, $\phi \vee \psi \sim \psi \vee \phi$. In `Agda` we can write the corresponding proof as follows:

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

Then, we show that the equivalence of the statements hold in `Agda`:

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

□

3.4.3 Associativity

Proposition 3. $\phi \wedge (\psi \wedge \gamma) \sim (\phi \wedge \psi) \wedge \gamma$

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 are very similar, we will only present the first one. However, the complete deduction tree is too large to include here, but can be found in A.1 for reference. In `Agda`, the corresponding proof of the first entailment is:

```
∧-assoc1 : ∀ {φ ψ γ : Formula} → Γ : φ ∧ (ψ ∧ γ) ⊢ (φ ∧ ψ) ∧ γ
∧-assoc1 = ∧-I (∧-I (∧-E₁ (axiom Z))
                    (∧-E₁ (∧-E₂ (axiom Z))))
                (∧-E₂ (∧-E₂ (axiom Z)))
```

The proof of the second entailment in `Agda`:

```
∧-assoc2 : ∀ {φ ψ γ : Formula} → Γ : (φ ∧ ψ) ∧ γ ⊢ φ ∧ (ψ ∧ γ)
∧-assoc2 = ∧-I (∧-E₁ (∧-E₁ (axiom Z)))
               (∧-I (∧-E₂ (∧-E₁ (axiom Z)))
                    (∧-E₂ (axiom Z)))
```

Therefore, we can conclude that the following equivalence holds:

```
ass-eq-∧ : ∀ {φ ψ γ : Formula} → φ ∧ (ψ ∧ γ) ~ (φ ∧ ψ) ∧ γ
ass-eq-∧ = ∧-assoc1 , ∧-assoc2
```

□

Proposition 4. $\phi \vee (\psi \vee \gamma) \sim (\phi \vee \psi) \vee \gamma$

Proof. We need to show $\Gamma, \phi \vee (\psi \vee \gamma) \vdash (\phi \vee \psi) \vee \gamma$ and $\Gamma, (\phi \vee \psi) \vee \gamma \vdash \phi \wedge (\psi \vee \gamma)$ Proof is done through natural deduction. The proofs are very similar and we have therefore opted to omitt the second proof, and we refer the reader to A.2 for details on the first one. In `Agda` we write the proofs as follows, the first entailment:

```
∨-assoc1 : ∀ {φ ψ γ : Formula} → Γ : φ ∨ (ψ ∨ γ) ⊢ (φ ∨ ψ) ∨ γ
∨-assoc1 = ∨-E (axiom Z)
               (∨-I₂ (∨-I₂ (axiom Z)))
               (∨-E (axiom Z)
                    (∨-I₂ (∨-I₁ (axiom Z)))
                    (∨-I₁ (axiom Z)))
```

The proof of the second entailment in `Agda`:

```
∨-assoc2 : ∀ {φ ψ γ : Formula} → Γ : (φ ∨ ψ) ∨ γ ⊢ φ ∨ (ψ ∨ γ)
∨-assoc2 = ∨-E (axiom Z)
               (∨-E (axiom Z)
                    (∨-I₂ (axiom Z))
                    (∨-I₁ (∨-I₂ (axiom Z))))
               (∨-I₁ (∨-I₁ (axiom Z)))
```

With these two results we have proven $\phi \vee (\psi \vee \gamma) \sim (\phi \vee \psi) \vee \gamma$. In `Agda`, the equivalence is shown as follows:

```
ass-eq-∨ : ∀ {φ ψ γ : Formula} → φ ∨ (ψ ∨ γ) ~ (φ ∨ ψ) ∨ γ
ass-eq-∨ = ∨-assoc1 , ∨-assoc2
```

□

3.4.4 Distributivity

Proposition 5. $\phi \wedge (\psi \vee \gamma) \sim (\phi \wedge \psi) \vee (\phi \wedge \gamma)$

Proof. We need to show $\Gamma, \phi \wedge (\psi \vee \gamma) \vdash (\phi \wedge \psi) \vee (\phi \wedge \gamma)$ and $\Gamma, (\phi \wedge \psi) \vee (\phi \wedge \gamma) \vdash \phi \wedge (\psi \vee \gamma)$. For the natural deductions, please refer to A.3 and A.4, together they show $\phi \wedge (\psi \vee \gamma) \sim (\phi \wedge \psi) \vee (\phi \wedge \gamma)$. The corresponding Agda proof of the first entailment is:

```
∧-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)))
```

In Agda, the second entailment is formalised as follows:

```
∧-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))))
```

We can combine these two proofs to show equivalence in Agda:

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

□

Proposition 6. $\phi \vee (\psi \wedge \gamma) \sim (\phi \vee \psi) \wedge (\phi \vee \gamma)$

Proof. We aim to show $\Gamma, \phi \vee (\psi \wedge \gamma) \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)$ and $\Gamma, (\phi \vee \psi) \wedge (\phi \vee \gamma) \vdash \phi \vee (\psi \wedge \gamma)$. For the deduction details please refer to A.5 and A.6. The following is the proof of the first entailment in Agda:

```
∨-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))))
```

The second entailment is shown in Agda as follows:

```
∨-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))))
```

We can thus conclude that the two statements are equivalent:

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

□

4 Lindenbaum-Tarski algebra in Cubical Agda

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 definiton of a set quotient from the `agda/cubical` library to define the Lindenbaum-Tarski algebra:

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

4.1 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:

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

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. The full proof in Agda:

```
∼-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 can define conjunction on Lindenbaum-Tarski algebra in `Agda` as follows:

```
_⋀_ : 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 ∧/, 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 defintion 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:

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

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 conjunction elimination we get the desired result:

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

Proving $\Gamma, a' \vee b' \vdash a \vee b$ is identical to the first proof, so we omit it here. In `Agda` the full proof is written as follows:

```
~-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 `Agda`:

```
_⋁_ : LindenbaumTarski → LindenbaumTarski → LindenbaumTarski
A ⋁ B = setQuotBinOp ~-refl ~-refl _∨_ ~-respects-∨ A B
```

Here, setQuotBinOp takes four arguments: two proofs of reflexivity (∼-refl) for the equivalence relation, the logical disjunction operator (∨/), and a proof that the operator respects the equivalence relation (∼-respects-∨). The function expects two reflexivity proofs, one for each of the two elements we are combining with the operator. In this case, both elements are of the same type, hence the reflexivity proofs are the same. The resulting function ∨/ takes two elements A and B in the Lindenbaum-Tarski algebra and returns their disjunction.

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 ∼. 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 ∼ 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 $\Gamma, \neg a \vdash \neg a'$. This is done through natural deduction:

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{\Gamma, a' \vdash a}{\Gamma, a', \neg a \vdash a}\,\textsc{weakening}
    }{\Gamma, \neg a, a' \vdash a}\,\textsc{exchange}
    \qquad
    \dfrac{
      \dfrac{\neg a \in \Gamma, \neg a}{\Gamma, \neg a \vdash \neg a}\,\textsc{axiom}
    }{\Gamma, \neg a, a' \vdash \neg a}\,\textsc{weakening}
  }{\Gamma, \neg a, a' \vdash \bot}\,\neg\text{-E}
}{\Gamma, \neg a \vdash \neg a'}\,\neg\text{-I}
$$

In Agda, we define the full proof as follows:

```
∼-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 ∼, and allows us to define negation on the Lindenbaum-Tarski algebra in Agda:

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

The `setQuotUnaryOp` function takes two arguments: a unary operator (¬/) and a proof that the operator respects the equivalence relation (∼-respects-¬). 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 quotient in `Cubical Agda`:

```
⊤/ : LindenbaumTarski
⊤/ = [ ⊤ ]
```

```
⊥/ : LindenbaumTarski
⊥/ = [ ⊥ ]
```

4.2 Proof that the Lindenbaum Tarski algebra is Boolean

Definition 4.2 (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 4.3. *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 4.4. *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 4.5. *A boolean algebra is a complemented distributive lattice.*

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

Proof. □

4.3 Soundness

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

A Derivations

A.1 Derivation of $\Gamma, \phi \wedge (\psi \wedge \gamma) \vdash (\phi \wedge \psi) \wedge \gamma$

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

$\mathcal{D}_1$

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

$\mathcal{D}_2$

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

A.2 Derivation of $\Gamma, \phi \vee (\psi \vee \gamma) \vdash (\phi \vee \psi) \vee \gamma$

$$\dfrac{\dfrac{\phi \vee (\psi \vee \gamma) \in \Gamma, \phi \vee (\psi \vee \gamma)}{\Gamma, \phi \vee (\psi \vee \gamma) \vdash \phi \vee (\psi \vee \gamma)}\ \textsc{axiom} \qquad \dfrac{\dfrac{\dfrac{\phi \in \Gamma, \phi \vee (\psi \vee \gamma), \phi}{\Gamma, \phi \vee (\psi \vee \gamma), \phi \vdash \phi}\ \textsc{axiom}}{\Gamma, \phi \vee (\psi \vee \gamma), \phi \vdash \phi \vee \psi}\ \vee\text{-I}_2}{\Gamma, \phi \vee (\psi \vee \gamma), \phi \vdash (\phi \vee \psi) \vee \gamma}\ \vee\text{-I}_2 \qquad \begin{matrix}\mathcal{D} \\ \Gamma, \phi \vee (\psi \vee \gamma), (\psi \vee \gamma) \vdash \psi \vee \gamma\end{matrix}}{\Gamma, \phi \vee (\psi \vee \gamma) \vdash (\phi \vee \psi) \vee \gamma}\ \vee\text{-E}$$

$\mathcal{D}$

$$\dfrac{\dfrac{\psi \vee \gamma \in \Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma}{\Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma \vdash \psi \vee \gamma}\ \textsc{axiom} \qquad \begin{matrix}\mathcal{D}' \\ \Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma, \psi \vdash (\phi \vee \psi) \vee \gamma\end{matrix} \qquad \dfrac{\dfrac{\gamma \in \Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma, \gamma}{\Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma, \gamma \vdash \gamma}\ \textsc{axiom}}{\Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma, \gamma \vdash (\phi \vee \psi) \vee \gamma}\ \vee\text{-I}_1}{\Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma \vdash (\phi \vee \psi) \vee \gamma}\ \vee\text{-E}$$

$\mathcal{D}'$

$$\dfrac{\dfrac{\dfrac{\psi \in \Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma, \psi}{\Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma, \psi \vdash \psi}\ \textsc{axiom}}{\Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma, \psi \vdash \phi \vee \psi}\ \vee\text{-I}_1}{\Gamma, \phi \vee (\psi \vee \gamma), \psi \vee \gamma, \psi \vdash (\phi \vee \psi) \vee \gamma}\ \vee\text{-I}_2$$

A.3 Derivation of $\Gamma, \phi \land (\psi \lor \gamma) \vdash (\phi \land \psi) \lor (\phi \land \gamma)$

$$\dfrac{\dfrac{\dfrac{\phi \land (\psi \lor \gamma) \in \Gamma, \phi \land (\psi \lor \gamma)}{\Gamma, \phi \land (\psi \lor \gamma) \vdash \phi \land (\psi \lor \gamma)}\,{\scriptstyle \text{AXIOM}}}{\Gamma, \phi \land (\psi \lor \gamma) \vdash \psi \lor \gamma}\,{\scriptstyle \land\text{-E}_2} \qquad \begin{matrix}\mathcal{D}_1 \\ \Gamma, \phi \land (\psi \lor \gamma), \psi \vdash (\phi \land \psi) \lor (\phi \land \gamma)\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Gamma, \phi \land (\psi \lor \gamma), \gamma \vdash (\phi \land \psi) \lor (\phi \land \gamma)\end{matrix}}{\Gamma, \phi \land (\psi \lor \gamma) \vdash (\phi \land \psi) \lor (\phi \land \gamma)}\,{\scriptstyle \lor\text{-E}}$$

$\mathcal{D}_1$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\phi \land (\psi \lor \gamma) \in \Gamma, \phi \land (\psi \lor \gamma)}{\Gamma, \phi \land (\psi \lor \gamma) \vdash \phi \land (\psi \lor \gamma)}\,{\scriptstyle \text{AXIOM}}}{\Gamma, \phi \land (\psi \lor \gamma) \vdash \phi}\,{\scriptstyle \land\text{-E}_1}}{\Gamma, \phi \land (\psi \lor \gamma), \psi \vdash \phi}\,{\scriptstyle \text{WEAKENING}} \qquad \dfrac{\psi \in \Gamma, \phi \land (\psi \lor \gamma), \psi}{\Gamma, \phi \land (\psi \lor \gamma), \psi \vdash \psi}\,{\scriptstyle \text{AXIOM}}}{\Gamma, \phi \land (\psi \lor \gamma), \psi \vdash \phi \land \psi}\,{\scriptstyle \land\text{-I}}}{\Gamma, \phi \land (\psi \lor \gamma), \psi \vdash (\phi \land \psi) \lor (\phi \land \gamma)}\,{\scriptstyle \lor\text{-I}_1}$$

$\mathcal{D}_2$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\phi \land (\psi \lor \gamma) \in \Gamma'}{\Gamma, \phi \land (\psi \lor \gamma) \vdash \phi \land (\psi \lor \gamma)}\,{\scriptstyle \text{AXIOM}}}{\Gamma, \phi \land (\psi \lor \gamma) \vdash \phi}\,{\scriptstyle \land\text{-E}_1}}{\Gamma, \phi \land (\psi \lor \gamma), \gamma \vdash \phi}\,{\scriptstyle \text{WEAKENING}} \qquad \dfrac{\gamma \in \Gamma, \phi \land (\psi \lor \gamma), \gamma}{\Gamma, \phi \land (\psi \lor \gamma), \gamma \vdash \gamma}\,{\scriptstyle \text{AXIOM}}}{\Gamma, \phi \land (\psi \lor \gamma), \gamma \vdash \phi \land \gamma}\,{\scriptstyle \land\text{-I}}}{\Gamma, \phi \land (\psi \lor \gamma), \gamma \vdash (\phi \land \psi) \lor (\phi \land \gamma)}\,{\scriptstyle \lor\text{-I}_1}$$

A.4 Derivation of $\Gamma, (\phi \land \psi) \lor (\phi \land \gamma) \vdash \phi \land (\psi \lor \gamma)$

Denote $\Gamma' = \Gamma \cup \{(\phi \land \psi) \lor (\phi \land \gamma)\}$

$$\dfrac{\begin{matrix}\mathcal{D}_1 \\ \Gamma' \vdash \phi\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Gamma' \vdash \psi \lor \gamma\end{matrix}}{\Gamma' \vdash \phi \land (\psi \lor \gamma)}\ \land\text{-I}$$

$\mathcal{D}_1$

$$\dfrac{\dfrac{(\phi \land \psi) \lor (\phi \land \gamma) \in \Gamma'}{\Gamma' \vdash (\phi \land \psi) \lor (\phi \land \gamma)}\ \textsc{axiom} \qquad \dfrac{\dfrac{\phi \land \psi \in \Gamma', \phi \land \psi}{\Gamma', \phi \land \psi \vdash \phi \land \psi}\ \textsc{axiom}}{\Gamma', \phi \land \psi \vdash \phi}\ \land\text{-E}_1 \qquad \dfrac{\dfrac{\phi \land \gamma \in \Gamma', \phi \land \gamma}{\Gamma', \phi \land \gamma \vdash \phi \land \gamma}\ \textsc{axiom}}{\Gamma', \phi \land \gamma \vdash \phi}\ \land\text{-E}_1}{\Gamma' \vdash \phi}\ \lor\text{-E}$$

$\mathcal{D}_2$

$$\dfrac{\dfrac{(\phi \land \psi) \lor (\phi \land \gamma) \in \Gamma'}{\Gamma' \vdash (\phi \land \psi) \lor (\phi \land \gamma)}\ \textsc{axiom} \qquad \dfrac{\dfrac{\dfrac{\phi \land \psi \in \Gamma', \phi \land \psi}{\Gamma', \phi \land \psi \vdash \phi \land \psi}\ \textsc{axiom}}{\Gamma', \phi \land \psi \vdash \psi}\ \land\text{-E}_2}{\Gamma', \phi \land \psi \vdash \psi \lor \gamma}\ \lor\text{-I}_2 \qquad \dfrac{\dfrac{\dfrac{\phi \land \gamma \in \Gamma', \phi \land \gamma}{\Gamma', \phi \land \gamma \vdash \phi \land \gamma}\ \textsc{axiom}}{\Gamma', \phi \land \gamma \vdash \gamma}\ \land\text{-E}_2}{\Gamma', \phi \land \gamma \vdash \psi \lor \gamma}\ \lor\text{-I}_1}{\Gamma' \vdash \psi \lor \gamma}\ \lor\text{-E}$$

A.5 Derivation of $\Gamma, \phi \vee (\psi \wedge \gamma) \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)$

$$\dfrac{\dfrac{\phi \vee (\psi \wedge \gamma) \in \Gamma, \phi \vee (\psi \wedge \gamma)}{\Gamma, \phi \vee (\psi \wedge \gamma) \vdash \phi \vee (\psi \wedge \gamma)}\,\textsc{axiom} \qquad \begin{matrix}\mathcal{D}_1\\ \Gamma, \phi \vee (\psi \wedge \gamma) \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)\end{matrix} \qquad \begin{matrix}\mathcal{D}_2\\ \Gamma, \phi \vee (\psi \wedge \gamma) \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)\end{matrix}}{\Gamma, \phi \vee (\psi \wedge \gamma) \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\,\vee\text{-E}$$

$\mathcal{D}_1$

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

$\mathcal{D}_2$

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

A.6 Derivation of $\Gamma, (\phi \vee \psi) \wedge (\phi \vee \gamma) \vdash \phi \vee (\psi \wedge \gamma)$

Denote $\Gamma' = \Gamma \cup \{(\phi \vee \psi) \wedge (\phi \vee \gamma)\}$

$$\dfrac{\dfrac{\dfrac{(\phi \vee \psi) \wedge (\phi \vee \gamma) \in \Gamma'}{\Gamma' \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\ \textsc{axiom}}{\Gamma' \vdash \phi \vee \psi}\ \wedge\text{-E}_1 \qquad \dfrac{\dfrac{\phi \in \Gamma', \phi}{\Gamma', \phi \vdash \phi}\ \textsc{axiom}}{\Gamma', \phi \vdash \phi \vee (\psi \wedge \gamma)}\ \vee\text{-I}_2 \qquad \begin{array}{c}\mathcal{D}\\ \Gamma', \psi \vdash \phi \vee (\psi \wedge \gamma)\end{array}}{\Gamma', \phi \vdash \phi \vee (\psi \wedge \gamma)}\ \vee\text{-E}$$

$\mathcal{D}$

$$\dfrac{\dfrac{\dfrac{\dfrac{(\phi \vee \psi) \wedge (\phi \vee \gamma) \in \Gamma'}{\Gamma' \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\ \textsc{axiom}}{\Gamma', \psi \vdash (\phi \vee \psi) \wedge (\phi \vee \gamma)}\ \textsc{weakening}}{\Gamma', \psi \vdash \phi \vee \gamma}\ \wedge\text{-E}_2 \qquad \dfrac{\dfrac{\phi \in \Gamma', \psi, \phi}{\Gamma', \psi, \phi \vdash \phi}\ \textsc{axiom}}{\Gamma', \psi, \phi \vdash \phi \vee (\psi \wedge \gamma)}\ \vee\text{-I}_2 \qquad \mathcal{D}'}{\Gamma', \psi, \phi \vdash \phi \vee (\psi \wedge \gamma)}\ \vee\text{-E}$$

$\mathcal{D}'$

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

B LindenbaumTarski.agda