SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

A proof searching procedure for intuitionistic propositional logic in Agda

av

Emma Bastås

2025 - No K8

A proof searching procedure for intuitionistic propositional logic in Agda

Emma Bastås

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Anders Mörtberg

2025

Acknowledements

Thank you Shin-Cheng Mu for your blog *niche computing science*, and Donnacha Oisín Kidney for sharing some hard-learned lessons about Agda online. Thank you to all of the people the world over who have contributed to the wealth of human knowlegde by asking and answering questions online, writing blog-posts, making tutorial videos, and sharing your passion for, and knowledge of computers, programming, logic, philosophy, and the intersection therein.

Thank you Charlie, Diddi, Eva, Lars-Johan, Leo, Molly, and Pernilla. I am incredibly privileged to have your support and love.

Thank you Anders Mörtberg, my thesis advisor, for having helped me find my way through this process, providing invaluable input, and sharing your experience.

Thank you David, my friend and colleague, for being by my side these last three years.

Abstract

Sequent calculus for intuitionistic propositional logic (IPC) is briefly presented and later formalized in Agda, an interactive theorem prover. The main result of this thesis is a proof searching procedure for IPC using sequent calculus, which is proven correct in Agda. We facilitate this in two ways; first by restricting the "normal" sequent calculus into a "contraction-free" version along the lines of R. Dyckhoff [1], guaranteeing termination of the search procedure. Secondly, the decision procedure itself is encoded in the inference rules, similarly to the "focused" sequent calculus of R. J. Simmons [2], leaving us with a slightly more complicated set of inference rules, but also a search procedure that is correct almost by construction. In the end, we end up with a sequent calculus LJf[1] for which a search procedure is implemented and proven correct in Agda.

Sammanfattning

Sekventkalkyl för intuitionistisk satslogik (IPC) presenteras kortfattat och formaliseras senare i Agda, en interaktiv bevisassistent. Uppsatsens huvudresultat är en bevissökningsalgoritm som bevisas vara korrekt i Agda. Vi möjliggör detta på två vis; först genom att begränsa den "vanliga" sekventkalkyllen till en "kontraktionsfri" version likt den formulerad av R. Dyckhoff [1], vilket garanterar att sökalgoritmen terminerar. För det andra så inskrivs algoritmens valprocedur i inferensreglerna själva, likt R. J. Simmons [2] "fokuserade" sekventkalkyl, vilket leder till inferensregler som är något mer komplicerade, men som gör att avgörbarhet håller nästan automatiskt. I slutändan får vi en sekventkalkyl LJf[2] för vilket en sökalgoritm implementeras och bevisas vara korrekt i Agda.

[1] LJf can be seen as a simpler, less efficient, variant of the focused sequent calculus LJF of Simmons, hence the lower-case f.

[2] LJf kan ses som en enklare, mindre effektiv, variant av Simmons fokuserade sekventkalkyl LJF, varav gemenen "f".

Contents

1. Introduction

The original formulation of sequent calculus is due to G. Gentzen [3] as a means of proving properties of natural deduction. He devised two variants of sequent calculus, one classical, denoted LK, and one intuitionistic, denoted LJ. Both LK and LJ can be used with propositional and predicate logic. In this thesis, we are going to focus on LJ for intuitionistic propositional logic (IPC) only, as presented by F. Pfenning [4]. Sequent calculus has since found other uses. One such use is for the development of mechanized proof searching, which is precisely what we are going to use it for. Another – and perhaps related – benefit of sequent calculus for us is that it is amenable to formalization in an interactive theorem prover such as Agda.

We are going to use Agda throughout this thesis, providing definitions, propositions and proofs in both pen-and-paper style and as Agda programs. The final Agda codebase is available at [5]. The reason for the use of Agda is that we are interested in developing a proof searching procedure *and* prove that this procedure is correct, and, by using Agda we can obtain both: In the end we will arrive with a function isProvable in Agda that *is* the search procedure, *and* a proof that the search procedure is correct. In Section 2 we will provide a brief overview of Agda and the "proofs as program" paradigm that underpins it and many other interactive theorem provers.

In Section 3.1 and Section 3.2 we will introduce LJ, and in Section 3.3 we formalize LJ in Agda. In Section 3.4 we will prove some basic properties of LJ in Agda, and in Section 3.5 we prove that LJ is sound in Agda with respect to classical semantics.

In Section 4 we begin by developing a restricted version of LJ denoted LJf, and then develop a search procedure for it.

In Section 5 we prove that the search procedure in Section 4 does terminate. Section 5.1 is concerned with how termination is proven in Agda using well-founded induction.

In Section 6 we show how a proof in LJf is translated to LJ.

2. About Agda

Agda is two things simultaneously. It is a programming language and a theorem prover, meaning that Agda can be used to both write computer programs and prove mathematical statements. It can be these two things at the same time because in Agda, every type is a proposition, and every program is a proof, $x : T$ is a proof of T, and $f : P \to Q$ is a proof that assuming P, Q can be proved. The phrases "propositions as types", "proofs as programs", "Curry-Howard isomorphism" are all related to this, though we will not explore this in detail [6]. The idea of interpreting types as propositions and programs as proofs can be applied to Haskell as well. In this interpretation, a function like $\mathsf{isMember} : \mathsf{Integer} \to \mathsf{List\ Integer} \to \mathsf{Bool}$ proves that given an integer, and a list of integers, there exists a boolean. This is not a very interesting proof, and we could write essentially the same function in Agda and have the same uninteresting interpretation. Things get interesting, however, when we leverage Agda's "dependent types", which lets us write a signature such as this

$$\mathsf{isProvable} : (\Gamma : \mathsf{Ctx}) \to (C : \mathsf{Prop}) \to \mathsf{Either}\ (\Gamma \vdash C)\ (\Gamma \nvdash C)$$

which is interpreted as a proof that, for a set of assumptions Γ and a propositional formula C, there is either a derivation in LJ assuming Γ and concluding C, or a proof that no such derivation exists in LJ. Proving this statement is the end goal of this thesis. $\Gamma \vdash C$ and $\Gamma \nvdash C$ are both types, and as we can see, the types refer back to the arguments Γ and C. This is not possible in Haskell, but it is in Agda, because Agda is dependently typed.

Now we introduce some more concepts in Agda that we will make use of in this thesis.

Proofs using Either

We saw a reference to an Either datatype, recall how Either is defined in Haskell

```
data Either a b = Left a | Right b
```

In Haskell-lingo, Either is a *type constructor*, meaning that it takes as arguments two types a and b and produces a new type Either a b. In Agda, type constructors can take vales as arguments, not only types, so the definition of Either is a little more explicit here than in Haskell

```
data Either (A : Set) (B : Set) : Set where
  left  : (x : A) → Either A B
  right : (x : B) → Either A B
```

The Either data type takes two arguments A and B of type Set and returns a Set. The Set type can be though of roughly as the "type of types"[3], that is, Either takes two types and returns a new type.

Truth and falsehood

In logic we usually have the atom $\top$ denoting a statement that is always true. In the types-as-propositions interpretation, this corresponds to a type that can always be constructed. We have this notion in both Agda and Haskell, and it is the unit type

```
data Unit : Set where
  tt : Unit
```

Falsehood $\bot$ should never be provable, and that corresponds to a type that can never be constructed.

```
data bot : Set where
```

Since bot has no constructors, we can write the following in order to encode the principle of explosion

```
bot-elim : ∀ {A : Set} → bot → A
bot-elim ()
```

The empty parenthesis is a so-called "absurd pattern," and we can pattern match using the absurd pattern when there are no other constructors applicable.

We also define negation of statements. We are used to thinking of the statement $\neg P$ as being true if and only if P is false, and in Agda this means that bot can be constructed by assuming P, giving us the following definition

[3]This is an oversimplification, in Agda there is a hierarchy of types denoted Set, Set₁, Set₂, ⋯, but that is beyond the scope of this thesis. See https://agda.readthedocs.io/en/v2.7.0.1/language/sort-system.html for futher information.

```
¬_ : Set → Set
¬ P = P → bot
```

Whenever statements such as x ≠ y, Γ ⊬ C, etc, are encountered in this thesis, they are always mere synonyms for x ≡ y → bot, etc.

Equality and Booleans

We will also make extensive use of propositional equality _≡_, refl as defined in Agda's standard library. If you haven't encountered these before I warmly recommend reading [7], especially about cong, cong₂ and subst as we will make use of these in this thesis.

The type of booleans is given the name 𝔹 in this thesis, with operators _or_, _and_, _cond_ and _not_.

2.1. Propositional formulas.

We define our set of propositions Prop in much the same way as [8].

Definition 2.1.1 (Prop): The set Prop of propositional formulas is defined inductively as follows:

$$\frac{n \in \mathbb{N}}{P_n \in \text{Prop}} \qquad \frac{}{\top \in \text{Prop}}$$

$$\frac{}{\bot \in \text{Prop}} \qquad \frac{A \in \text{Prop} \quad B \in \text{Prop}}{(A \wedge B) \in \text{Prop}}$$

$$\frac{A \in \text{Prop} \quad B \in \text{Prop}}{(A \vee B) \in \text{Prop}} \qquad \frac{A \in \text{Prop} \quad B \in \text{Prop}}{(A \to B) \in \text{Prop}}$$

This is how we formalize it in Agda.

```
data Prop : Set where
  Pvar : ℕ → Prop
  ⊤    : Prop
  ⊥    : Prop
  _∧_  : Prop → Prop → Prop
  _∨_  : Prop → Prop → Prop
  _⟶_  : Prop → Prop → Prop
```

(Note that → and ⟶ look similar but are distinct symbols. The first is an Agda keyword used in function signatures and the latter is a constructor.)

We have said nothing about two common logical connectives: negation $\neg$, and biconditional $\leftrightarrow$. This is simply because they can be described in terms of the other logical connectives:

$$\underbrace{\neg A :\equiv A \to \bot}_{\text{see the similarities to Agda?}} \qquad A \leftrightarrow B :\equiv (A \to B) \wedge (B \to A),$$

and so we omit them for brevity.

2.2. Contexts and context membership

Here we make precise the notion of a context, which is used to define sequents in Section 3.1.

A context is roughly a set containing all propositions we are allowed to assume in a proof. I say "roughly" because, in the literature, contexts are usually conceived of as a multisets (a set where there can be multiple occurrences of the same element), and we will follow this convention. For the Agda formalization, however, contexts are list-like (for purely practical reasons), and order does matter, but Proposition 3.4.2.1 will show that the multiset and list notions are in some sense equivalent.

We define the Ctx type in Agda, along with the definition for concatenation $+\!\!+$.

```
data Ctx : Set where
  ∅   : Ctx
  _,_ : Ctx → Prop → Ctx

_++_ : Ctx -> Ctx -> Ctx
∅ ++ Δ = Δ
(Γ , x) ++ Δ = (Γ ++ Δ) , x
```

Example: The context z, y, x is ∅ , z , y , x in Agda. Of note is that the head of the context, x, if furthest to the right, while the head is usually furthest to the left in many programming language contexts.

Now we need a notion of context membership. Let x ∈ Γ be the type of proofs that the proposition x is in the context Γ, we define this relation as follows

```
data _∈_ : Prop → Ctx → Set where
  head : {e : Prop} →                {Γ : Ctx} → e ∈ Γ , e
  tail : {e : Prop} → {x : Prop} {Γ : Ctx} → e ∈ Γ → e ∈ Γ , x
```

To better understand the idea behind this datatype, it might be instructive to consider the constructors head and tail as if they were inference rules.

$$\frac{}{e \in \Gamma, e}\,\text{head} \qquad \frac{e \in \Gamma}{e \in \Gamma, x}\,\text{tail}$$

The "head" rule states that e is always an element in the context Γ, e, because e is at the head of the context. The "tail" rule states that if we know that e is in Γ then e is also in Γ, x, because e is in the tail of the context.

Example: The statement $x \in \emptyset, z, y, x$ is proved using inference rules and Agda as follows

$$\frac{}{x \in \emptyset, z, y, x}\,\text{head}$$

```
ex₁ : ∀ {x y z} → x ∈ ∅ , z , y , x
ex₁ = head
```

Example: The statement $z \in \emptyset, z, y, x$ is proved using inference rules and Agda as follows

$$\dfrac{\dfrac{\dfrac{}{z \in \emptyset, z}\,\text{head}}{z \in \emptyset, z, y}\,\text{tail}}{z \in \emptyset, z, y, x}\,\text{tail}$$

```
ex₂ : ∀ {x y z} → z ∈ ∅ , z , y , x
ex₂ = tail (tail head)
```

3. Sequent calculus

In this section we first define intuitionistic sequent calculus LJ for IPC along the lines of F. Pfenning [4]. In Section 3.3 we formalize LJ in Agda.

3.1. The syntax of sequent calculus

A *sequent* in LJ is a sentence of the form

$$A_1, \cdots, A_n \vdash C$$

Where $A_1, \cdots, A_n$ and C are all elements of Prop. We can read this sentence as stating "If we can assume $A_1, \cdots, A_n$ then we may conclude C". We call the assumptions $A_1, \cdots, A_n$ the *antecendents* or *context*, and C the *succedent*.

> *Example*: $P_1 \wedge P_2 \vdash P_1 \vee P_3$ is a sequent stating that if $P_1 \wedge P_2$ is true, then so too must $P_1 \vee P_3$ be. This statement is something the we intuitively think of as being a true statement. In contrast, the sequent $P_1 \vdash P_2$ is something we judge intuitively to be a false statement. So there are both "good" and "bad" sequents, and what we want to do is develop a set of rules that lets us construct only the "good" ones.

Of note is that the order of the *antecendents* is irrelevant, and it's also convenient to use notation like

$$\Gamma, A \vdash C$$

where Γ denotes some arbitrary set of antecendents the we are not interested in at this moment. The above sentence reads as "If we can assume A, and some other arbitrary (there might also be none at all) antecendents contained in Γ then we may conclude C".

3.2. Inference rules

We are only interested in sequents that state true things. To this end, we define the set LJ which contains all the sequents we believe state true things (this notion is made precise and proved in the proof of soundness). This set is defined inductively by a set of inference rules. We will introduce these inference rules now.

Truth.

The propositional atom $\top$ represents a statement that is always true, and as such we may always state it and be confident that we are right. So, in other words, for any set of assumptions in Γ, the sequent $\Gamma \vdash \top$ is always something we belive to be true, and so it should be in the set LJ. We summarize this with the inference rule

$$\dfrac{}{\Gamma \vdash \top}\,\top_R$$

Propositional variables.

The propositional atom P_n represents some statement that is true depending on how it is interpreted, and so we cannot be as bold as we were with $\top$ and simply proclaim "P_n is true", because there are certainly some interpretations where this statement is true, for instance if we interpret P_n to be "Plato is a man", but there are of course other interpretations in which the statement becomes false. But when can we be sure that P_n is true? Well, if we simply assume that P_n is true, then it is certainly true. In other words, for any set of assumptions Γ where P_n is a member, the sequent $\Gamma \vdash P_n$ is always something we believe to be true. We can summarize this with the inference rule

$$\frac{}{\Gamma, P_n \vdash P_n}\,\mathrm{id}$$

Conjunction.

If A and B are propositions, when can we be confident that $A \wedge B$ is a true statement? Well, whenever we are confident that both A is true and that B is true. We can summarize this with the inference rule

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge_{\mathrm{R}}$$

There are two more inference rules concerning conjunctions, but these are about manipulating the context Γ. They look like this

$$\frac{\Gamma, A \wedge B, A \vdash C}{\Gamma, A \wedge B \vdash C}\,\wedge_{\mathrm{L}_1} \qquad \frac{\Gamma, A \wedge B, B \vdash C}{\Gamma, A \wedge B \vdash C}\,\wedge_{\mathrm{L}_2}$$

They mean simply that if we have assumed $A \wedge B$, then we may also assume A and B if we want to.

Remark (L and R): The first of these rules, $\wedge_{\mathrm{R}}$ is called a right-rule and the other two, $\wedge_{\mathrm{L}_1}$ and $\wedge_{\mathrm{L}_2}$ are called left-rules. We will see that the inference rules for disjunction and implication also follow this scheme. The left/right distinction tells us which rules concern the left resp. the right side of the sequent we are trying to prove. With $\wedge_{\mathrm{R}}$, we are in some sense "looking" at the succedent of $\Gamma \vdash A \wedge B$ and deciding to act on the fact that it's a conjunction. With the left rules, we are instead "looking" at one of the antecendents.

Disjunction.

We begin with the two right rules for disjunction

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee_{\mathrm{R}_1} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee_{\mathrm{R}_2}$$

These reflect the idea that if we want to prove $A \vee B$, then it is enough tho show that either A or B holds.

The left rule is a little more complicated

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C} \vee_{\mathrm{L}}$$

The rationale for this one is that if we want to prove that assuming Γ and $A \vee B$ then C holds, then we find ourselves a little annoyed because we know that $A \vee B$ holds, but we cannot know if it holds because A is true, of because B is true, it could be either one of them. But, if we can show that in either case C holds, then we are done.

Implication.

The right rule is simple: If we want to prove that assuming Γ then $A \to B$ holds. Well, we could simply assume Γ and A, and show that B holds.

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to_{\mathrm{R}}$$

The left rule is more complicated.

$$\frac{\Gamma, A \to B \vdash A \quad \Gamma, A \to B, B \vdash C}{\Gamma, A \to B \vdash C} \to_{\mathrm{L}}$$

The idea is that if we have assumed $A \to B$, then, if we can prove A, we may assume B in addition to $A \to B$. The first premise of the inference rule is the proof of A, and the second premise is the proof of C where B is assumed.

Falsehood.

What can be said about the propositional atom $\bot$ that represents a statement that is always false? Well, since it is false we should never be able to conclude that $\bot$ is true, i.e. there should be no right rule for the sequent $\Gamma \vdash \bot$.

Moreover, we have the principle of explosion, or *Ex falso quodlibet*, which states that if we assume something false, then anything, even false things, can be proved. We specify this principle with the following left rule

$$\frac{}{\Gamma, \bot \vdash C} \bot_{\mathrm{L}_2}$$

Cut.

This is special rule as it does not pertain to any specific propositional atom or connective. It looks like this

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C} \text{cut}$$

This can be thought of as a analogy to the use of a lemma in normal mathematical reasoning: If I'm trying to prove C assuming Γ, and some lemma states that A holds assuming Γ, then I can additionally assume A in my proof of C.

The cut rule is also an example of an "admissible" rule, meaning that we can prove the statement

$$\text{If} \quad \Gamma \vdash A \quad \text{and} \quad \Gamma, A \vdash C \quad \text{then} \quad \Gamma \vdash C$$

for LJ using only the inference rules we have already defined. The fact that cut is admissible is a very important property, and Gentzen, the originator of sequent calculus even called the theorem the "Hauptsatz" (main theorem) [9]. Admissibility of cut is important to us too, and the restricted sequent calculus presented in Section 4.1 will not have cut rule, the reason is that it would be detrimental to a search procedure: If we want to search for a proof tree of $\Gamma \vdash C$ using cut, then we need to first find a proof tree of $\Gamma \vdash A$ for some A, but here A can be any proposition, which one should we pick? In some sense, the cut rule would offer to much room for creativity in a proof search, something we don't want.

Now we have defined all our inference rules, which we summarize below.

$$\frac{}{\Gamma, P_n \vdash P_n}\,\mathrm{id} \qquad \frac{}{\Gamma \vdash \top}\,\top_{\mathrm{R}} \qquad \frac{}{\Gamma, \bot \vdash C}\,\bot_{\mathrm{L}_2}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge_{\mathrm{R}} \qquad \frac{\Gamma, A \wedge B, A \vdash C}{\Gamma, A \wedge B \vdash C}\,\wedge_{\mathrm{L}_1} \quad \frac{\Gamma, A \wedge B, B \vdash C}{\Gamma, A \wedge B \vdash C}\,\wedge_{\mathrm{L}_2}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee_{\mathrm{R}_1} \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee_{\mathrm{R}_2} \qquad \frac{\Gamma, A \vee B, A \vdash C \quad \Gamma, A \vee B, B \vdash C}{\Gamma, A \vee B \vdash C}\,\vee_{\mathrm{L}}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\to_{\mathrm{R}} \qquad \frac{\Gamma, A \to B \vdash A \quad \Gamma, A \to B, B \vdash C}{\Gamma, A \to B \vdash C}\,\to_{\mathrm{L}}$$

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C}\,\mathrm{cut}$$

Figure 1: The set LJ defined with inference rules.

3.3. Sequent calculus in Agda

Now it's time to model LJ in Agda, but creating a new datatype with constructors based on the inference rules in Figure 1 will prove difficult as there are some ambiguities that us humans barely notice but that Agda will not accept. Let's illustrate this with an example. Here is a proof tree of the sequent $\Gamma, P_1 \wedge P_2, A \wedge B \vdash P_1 \wedge P_2$ in LJ

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, P_1 \wedge P_2, A \wedge B, P_2, P_1 \vdash P_1}\,\mathrm{id} \quad \dfrac{}{\Gamma, P_1 \wedge P_2, A \wedge B, P_2, P_1 \vdash P_2}\,\mathrm{id}}{\Gamma, P_1 \wedge P_2, A \wedge B, P_2, P_1 \vdash P_1 \wedge P_2}\,\wedge_{\mathrm{R}}}{\Gamma, P_1 \wedge P_2, A \wedge B, P_2 \vdash P_1 \wedge P_2}\,\wedge_{\mathrm{L}_1}}{\Gamma, P_1 \wedge P_2, A \wedge B \vdash P_1 \wedge P_2}\,\wedge_{\mathrm{L}_2}$$

Now we may ask ourselves which propositions in the context $\wedge_{\mathrm{L}_1}$ and $\wedge_{\mathrm{L}_2}$ refer to. The obvious answer to us humans is $P_1 \wedge P_2$, but to Agda that would not at all be clear: why couldn't it refer to $A \wedge B$ instead? Or some other conjunction in Γ? In Agda, for every left rule, we need to attach some more information. A first thought would be to attach some sort of index to our left rules specifying what proposition in the context we're referring to. The proof tree would then change to look like this

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, P_1 \wedge P_2, A \wedge B, P_2 \vdash P_1 \wedge P_2}\,[3]\ \wedge_{\mathrm{L}_1}}{\Gamma, P_1 \wedge P_2, A \wedge B \vdash P_1 \wedge P_2}\,[2]\ \wedge_{\mathrm{L}_2}$$

It will still be difficult to make Agda happy with this approach, though, we need something better. Instead of supplying an index we supply a proof that the proposition we refer to is an element in the context using the $\in$ datatype defined in Section 2.2. This works because the proof of x $\in$ xs contains three bits of information that Agda needs:

1. What the proposition x we are referring to looks like.
2. That the proposition x really is in the list xs.
3. Where precisely in xs the proposition x is located.

So we would change the $\wedge_{\mathrm{L}_1}$ and $\wedge_{\mathrm{L}_2}$ rules to look like this

$$\dfrac{A \wedge B \in \Gamma \quad \Gamma, A \vdash C}{\Gamma \vdash C}\,\wedge_{\mathrm{L}_1} \qquad\qquad \dfrac{A \wedge B \in \Gamma \quad \Gamma, B \vdash C}{\Gamma \vdash C}\,\wedge_{\mathrm{L}_2}$$

The id rule also needs to change. We now provide an explicit proof that the propositional variable that we want to use is in the context

$$\dfrac{P_n \in \Gamma}{\Gamma \vdash P_n}\,\mathrm{id}$$

Thus, our proof for the sequent $\Gamma, P_1 \wedge P_2, A \wedge B \vdash P_1 \wedge P_2$ changes to the following, which is quite verbose, maybe even painful, for us humans, but just right for Agda

$$\dfrac{\dfrac{\dfrac{}{P_1 \wedge P_2 \in \Gamma, P_1 \wedge P_2}\,\text{head}}{P_1 \wedge P_2 \in \Gamma, P_1 \wedge P_2, A \wedge B}\,\text{tail}}{\underbrace{\Gamma, P_1 \wedge P_2, A \wedge B}_{\Delta} \vdash P_1 \wedge P_2}\,\wedge_{L_2} \qquad \dfrac{\dfrac{\dfrac{}{P_1 \wedge P_2 \in \Gamma, P_1 \wedge P_2}\,\text{head}}{P_1 \wedge P_2 \in \Gamma, P_1 \wedge P_2, A \wedge B}\,\text{tail} \quad \dfrac{\dfrac{}{P_1 \in \Delta, P_2, P_1}\,\text{head}}{\Delta, P_2, P_1 \vdash P_1}\,\text{id} \quad \dfrac{\dfrac{\dfrac{}{P_2 \in \Delta, P_2}\,\text{head}}{P_2 \in \Delta, P_2, P_1}\,\text{tail}}{\Delta, P_2, P_1 \vdash P_2}\,\text{id}}{\Delta, P_2, P_1 \vdash P_1 \wedge P_2}\,\wedge_R}{P_1 \wedge P_2 \in \Gamma, P_1 \wedge P_2, A \wedge B, P_2}\,\text{tail} \qquad \Delta, P_2 \vdash P_1 \wedge P_2}\,\wedge_{L_1}$$

We institute this change for all of the other left rules as well, obtaining a new set of inference rules.

$$\frac{P_n \in \Gamma}{\Gamma \vdash P_n}\,\text{id} \qquad \frac{}{\Gamma \vdash \top}\,\top_R \qquad \frac{\bot \in \Gamma}{\Gamma \vdash C}\,\bot_{L_2}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge_R \qquad \frac{A \wedge B \in \Gamma \quad \Gamma, A \vdash C}{\Gamma \vdash C}\,\wedge_{L_1} \qquad \frac{A \wedge B \in \Gamma \quad \Gamma, B \vdash C}{\Gamma \vdash C}\,\wedge_{L_2}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee_{R_1} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee_{R_2} \qquad \frac{A \vee B \in \Gamma \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, \vdash C}\,\vee_L$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\to_R \qquad \frac{A \to B \in \Gamma \quad \Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma \vdash C}\,\to_L$$

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C}\,\text{cut}$$

Figure 2: The set LJ defined with more verbose inference rules.

These inference rules map to constructors in a new Agda datatype $\vdash$.

```
data _⊢_ : Ctx → Prop → Set₁ where
  id  : ∀ {Γ} {n}
      → Pvar n ∈ Γ
        ---------- id
      → Γ ⊢ Pvar n

  ⊤R : ∀ {Γ}
       ------ ⊤R
      → Γ ⊢ ⊤

  ⊥L : ∀ {Γ} {C}
      → ⊥ ∈ Γ
        ------ ⊥L
      → Γ ⊢ C

  ∧R : ∀ {Γ} {A B}
      → Γ ⊢ A
      → Γ ⊢ B
        ---------- ∧R
      → Γ ⊢ A ∧ B

  ∧L₁ : ∀ {Γ} {A B C}
      → A ∧ B ∈ Γ
      → Γ , A ⊢ C
        ---------- ∧L₁
      → Γ ⊢ C

  ∧L₂ : ∀ {Γ} {A B C}
      → A ∧ B ∈ Γ
      → Γ , B ⊢ C
        ---------- ∧L₂
      → Γ ⊢ C

  ∨R₁ : ∀ {Γ} {A B}
      → Γ ⊢ A
        ---------- ∨R₁
      → Γ ⊢ A ∨ B

  ∨R₂ : ∀ {Γ} {A B}
      → Γ ⊢ B
        ---------- ∨R₂
      → Γ ⊢ A ∨ B

  ∨L  : ∀ {Γ} {A B C}
      → A ∨ B ∈ Γ
      → Γ , A ⊢ C
      → Γ , B ⊢ C
        ---------- ∨L
      → Γ ⊢ C

  →R : ∀ {Γ} {A B}
      → Γ , A ⊢ B
        ---------- →R
      → Γ ⊢ A ⟶ B

  →L : ∀ {Γ} {A B C}
      → A ⟶ B ∈ Γ
      → Γ ⊢ A
      → Γ , B ⊢ C
        ---------- →L
      → Γ ⊢ C

  cut : ∀ {Γ} {A C}
      → Γ ⊢ A
      → Γ , A ⊢ C
        ---------- cut
      → Γ ⊢ C
```

Figure 3: The set LJ defined in Agda

Example: The sequent $P_1 \wedge P_2 \vdash P_2 \wedge P_1$ is proved using the inference rules in Figure 1 and Figure 3 as follows.

$$\dfrac{\dfrac{\dfrac{}{P_1 \wedge P_2, P_2 \vdash P_2}\,\mathrm{id}}{P_1 \wedge P_2 \vdash P_2}\,\wedge_{\mathrm{L}_2} \qquad \dfrac{\dfrac{}{P_1 \wedge P_2, P_1 \vdash P_1}\,\mathrm{id}}{P_1 \wedge P_2 \vdash P_2}\,\wedge_{\mathrm{L}_1}}{P_1 \wedge P_2 \vdash P_2 \wedge P_2}\,\wedge_{\mathrm{R}}$$

```
ex₁ : ∅ , P₁ ∧ P₂ ⊢ P₂ ∧ P₁
ex₁ = ∧R
  (∧L₂ head (id head))
  (∧L₁ head (id head))
```

Example: The sequent $((A \to \bot) \to \bot) \to \bot \vdash A \to \bot$ is proved using the inference rules in Figure 1 and Figure 3 as follows.

$$\dfrac{\dfrac{\dfrac{}{((A \to \bot) \to \bot) \to \bot, A, (A \to \bot) \vdash A}\,\mathrm{id} \quad \dfrac{}{((A \to \bot) \to \bot) \to \bot, A, (A \to \bot), \bot \vdash \bot}\,\bot_{L_2}}{((A \to \bot) \to \bot) \to \bot, A, (A \to \bot) \vdash \bot}\,{\to_L} \quad \dfrac{}{((A \to \bot) \to \bot) \to \bot, A, \bot \vdash \bot}\,\bot_{L_2}}{((A \to \bot) \to \bot) \to \bot, A \vdash (A \to \bot) \to \bot}\,{\to_R} \quad {\to_L}}{((A \to \bot) \to \bot) \to \bot, A \vdash \bot}\,{\wedge_R}$$

$$((A \to \bot) \to \bot) \to \bot \vdash A \to \bot$$

```
ex₂ : ∅ , ((P₁ ⟶ ⊥) ⟶ ⊥) ⟶ ⊥ ⊢ P₁ ⟶ ⊥
ex₂ = →R (→L
  (tail head)
  (→R (→L
    head
    (id (tail head))
    (⊥L head)))
  (⊥L head))
```

Now that we have LJ formalized in Agda, we want to prove a couple of useful properties of it, working our way towards a proof of soundness.

3.4. Structural rules.

Here we prove some useful properties of sequent calculus that we will make use of later, and also get more acquainted with proofs in Agda. The properties we prove are very basic, and also serve as some sort of sanity-check: if our formalization of LJ didn't have these properties then something might have gone wrong.

3.4.1. Strong identity

The strong-identity rule is like the id rule, except it holds for all propositions, not just propositional variables.

Proposition 3.4.1.1 (Strong identity): For all propositions A and all contexts Γ,

$$\Gamma, A \vdash A.$$

Equivalently

$$\text{If} \quad A \in \Gamma \quad \text{then} \quad \Gamma \vdash A.$$

We formulate this in Agda as

```
strong-id : ∀ {Γ} {A} → A ∈ Γ → Γ ⊢ A
```

Proof: Suppose that $A \in \Gamma$, and that there is a proof h for this using the head and tail inference rules define in Section 2.2. We proceed structural induction on A.

Base case 1 : $A = P_n$

This is true by the id rule. In Agda the case looks like this

```
strong-id {_} {Pvar n} h = id h
```

Base case 2 and 3 : $A = \top$ and $A = \bot$

These cases are trivially true just like case 1.

Inductive case 4 : $A = A_1 \wedge A_2$

By the inductive hypothesis we have proof trees of both $\Gamma, A_1 \vdash A_1$ and $\Gamma, A_2 \vdash A_2$. We thus obtain the following proof tree for $\Gamma \vdash A_1 \wedge A_2$

$$\dfrac{\dfrac{h}{A_1 \wedge A_2 \in \Gamma} \quad \dfrac{\dfrac{}{A_1 \in \Gamma, A_1}\,\text{head}}{\Gamma, A_1 \vdash A_1}\,\text{strong-id}}{\Gamma \vdash A_1}\,\wedge_{L_1} \qquad \dfrac{\dfrac{h}{A_1 \wedge A_2 \in \Gamma} \quad \dfrac{\dfrac{}{A_2 \in \Gamma, A_2}\,\text{head}}{\Gamma, A_2 \vdash A_2}\,\text{strong-id}}{\Gamma \vdash A_2}\,\wedge_{L_2}}{\Gamma \vdash A_1 \wedge A_2}\,\wedge_R$$

In Agda the case is handled like this

```
strong-id {_} {A₁ ∧ A₂} h =
  ∧R
    (∧L₁ h (strong-id head))
    (∧L₂ h (strong-id head))
```

Inductive case 5 and 6 : $A = A_1 \vee A_2$ and $A = A_1 \to A_2$

These cases are handled analogous to case 4, using the $\vee_L$, $\vee_{R_1}$ and $\vee_{R_2}$ rules resp. the $\to_R$ and $\to_L$ rules.

□

3.4.2. Strong weakening

Here we will present and sketch out a proof of "strong weakening," which is a generalization of three usual properties of sequent calculus called weakening, contraction and exchange.

Weakening stems from the observation that if we have assumed Γ, and proved C, then there is no reason why we can't add some extra, superfluous assumption A, and then assume Γ, A and prove C. That is

$$\text{If} \quad \Gamma \vdash C \quad \text{then} \quad \Gamma, A \vdash C \qquad \forall A \in \text{Prop}.$$

This is weakening. For strong weakening we first define a subset relation for contexts.

Definition 3.4.2.1 (Context subset): For two contexts Γ and Δ we say that Γ is a subset of Δ, denoted $\Gamma \subseteq \Delta$ if and only if

$$x \in \Gamma \Rightarrow x \in \Delta,$$

that is, every assumption in Γ is also an assumption in Δ.

In Agda this relation is defined as

```
_⊆_ : Ctx → Ctx → Set
_⊆_ Γ Δ = ∀ {x} → x ∈ Γ → x ∈ Δ
```

So in Agda, the way we prove that $\Gamma \subseteq \Delta$ is by producing a function that takes proofs of $x \in \Gamma$ and returns proofs of $x \in \Delta$.

Remark: This notion of a subset relation for contexts has some seemingly weird consequences, for instance, the context Γ, x, x, x is a subset of the context Γ, x. However, this subset relation still retains many properties that we want a subset relation to have, for instance the relation is still reflexive and transitive.

We also state a lemma that we will make use of.

Lemma 3.4.2.1: Adding assumptions to contexts is monotonic w.r.t. the subset relation, that is

$$\text{If} \quad \Gamma \subseteq \Delta \quad \text{then} \quad \Gamma, x \subseteq \Delta, x.$$

In Agda we state this as

```
,-mono-⊆ : ∀ {Γ Δ : Ctx} {x} → Γ ⊆ Δ → Γ , x ⊆ Δ , x
```

Proposition 3.4.2.1 (Strong weakening): Let Γ and Δ be contexts such that $\Gamma \subseteq \Delta$. If $\Gamma \vdash C$ then $\Delta \vdash C$.

In Agda we state this as

```
strong-weaken : ∀ {Γ Δ} {C} → Γ ⊆ Δ → Γ ⊢ C → Δ ⊢ C
```

Proof: Let $\Gamma \subseteq \Delta$ and proceed by structural induction on the proof tree $\mathcal{D}$ of the sequent $\Gamma \vdash C$.

Base case 1, 2 and 3 :

If the proof tree $\mathcal{D}$ is one of

$$\frac{P_n \in \Gamma}{\Gamma \vdash P_n}\,\mathrm{id} \qquad \frac{}{\Gamma \vdash \top}\,\mathrm{id} \qquad \frac{\bot \in \Gamma}{\Gamma \vdash C}\,\mathrm{id}$$

then the statement is trivially true, having to note only that since $\Gamma \subseteq \Delta$ we have $P_n \in \Delta$ resp. $\bot \in \Delta$. In Agda the cases are handled like this

```
strong-weaken Γ⊆Δ (id ∈Γ)   = id   (Γ⊆Δ ∈Γ)
strong-weaken Γ⊆Δ ⊤R        = ⊤R
strong-weaken Γ⊆Δ (⊥L ∈Γ) = ⊥L (Γ⊆Δ ∈Γ)
```

Inductive case 5 : $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A} \qquad \overset{\mathcal{D}_2}{\Gamma, A \vdash C}}{\Gamma \vdash C}\,\mathrm{cut}$

By the inductive hypothesis on $\mathcal{D}_1$ and $\mathcal{D}_2$ we have proof trees $\mathcal{E}_1$ and $\mathcal{E}_2$ of

$$\Delta \vdash A \quad \text{and} \quad \Delta, A \vdash C.$$

Now we construct a proof tree of $\Delta \vdash C$ as follows

$$\frac{\overset{\mathcal{E}_1}{\Delta \vdash A} \qquad \overset{\mathcal{E}_2}{\Delta, A \vdash C}}{\Delta \vdash C}\,\mathrm{cut}$$

In Agda the case is handled like this

```
strong-weaken Γ⊆Δ (cut 𝒟₁ 𝒟₂) =
  cut
    (strong-weaken Γ⊆Δ 𝒟₁)
    (strong-weaken
      (,-mono-⊆ Γ⊆Δ)
      𝒟₂
    )
```

Inductive case 5 : $\mathcal{D} = \dfrac{\overset{h}{A \vee B \in \Gamma} \qquad \overset{\mathcal{D}_1}{\Gamma, A \vdash C} \qquad \overset{\mathcal{D}_2}{\Gamma, B \vdash C}}{\Gamma \vdash C}\,\vee_{\mathrm{L}}$

We note first that $\Gamma, A \subseteq \Delta, A$ and $\Gamma, B \subseteq \Delta, B$ by Lemma 3.4.2.1. By the inductive hypothesis on $\mathcal{D}_1$ and $\mathcal{D}_2$ we have proof trees $\mathcal{E}_1$ and $\mathcal{E}_2$ of

$$\Delta, A \vdash C \quad \text{and} \quad \Delta, B \vdash C.$$

Finally, since we know that $A \vee B \in \Gamma$, and $\Gamma \subseteq \Delta$ we conclude that $A \vee B \in \Delta$ and label this proof h'.

Now we can construct a proof tree of $\Delta \vdash C$

$$\frac{A \vee B \in \Delta^{h'} \quad \Delta, A \vdash C^{\mathcal{E}_1} \quad \Delta, B \vdash C^{\mathcal{E}_2}}{\Delta \vdash C} \vee_{\mathrm{L}}$$

In Agda this case is handled as follows

```
strong-weaken Γ⊆Δ (∨L h 𝒟₁ 𝒟₂) =
  ∨L
    (Γ⊆Δ h)
    (strong-weaken (,-mono-⊆ Γ⊆Δ) 𝒟₁)
    (strong-weaken (,-mono-⊆ Γ⊆Δ) 𝒟₂)
```

The rest of the cases are handled analogously, using the inductive hypothesis on each subterm and using the assumption $\Gamma \subseteq \Delta$ and Lemma 3.4.2.1 where applicable. □

Now we state the three corollaries of weakening, contraction and exchange.

Corollary *(Weakening)*:

```
weaken : ∀ {Γ : Ctx} {x C} → Γ ⊢ C → Γ , x ⊢ C
```

Corollary *(Contraction)*:

```
contract : ∀ {Γ : Ctx} {x C} → x ∈ Γ → Γ , x ⊢ C → Γ ⊢ C
```

Corollary *(Exchange)*:

```
exchange : ∀ {Γ : Ctx} {A B C} → Γ , A , B ⊢ C → Γ , B , A ⊢ C
```

3.5. Soundness

Here we prove that our formalization of LJ is sound with respect to boolean semantics. Boolean semantics is a semantics of classical logic, while the sequent calculus we have developed is intuitionistic. The consequence of this is, for instance, that the proposition $A \vee \neg A$, is a tautology in the semantics, but not provable in LJ. However, this is only a problem when proving completeness, which we will not do in this thesis, and therefore we stick to boolean semantics because it might be more familiar to the reader. We define valuations along the lines of [8].

Definition 3.5.1 (Valuation): The valuation of the proposition A according to an assignment $a : \mathbb{N} \to \{\text{true}, \text{false}\}$, denoted $[\![A]\!]^a$ is given recursively by:

$$
\begin{aligned}
[\![P_i]\!]^a &= a(i) \\
[\![\top]\!]^a &= \text{true} \\
[\![\bot]\!]^a &= \text{false} \\
[\![A \wedge B]\!]^a &= [\![A]\!]^a \wedge_b [\![B]\!]^a \\
[\![A \vee B]\!]^a &= [\![A]\!]^a \vee_b [\![B]\!]^a \\
[\![A \to B]\!]^a &= [\![A]\!]^a \underset{b}{\to} [\![B]\!]^a
\end{aligned}
$$

where $\wedge_b$, $\vee_b$ and $\underset{b}{\to}$ denote the usual boolean operators.

The valuation $[\![\cdot]\!]$ is thus a function from $\text{Prop} \times (\text{Prop} \to \{\text{true}, \text{false}\})$ to $\{\text{true}, \text{false}\}$ where $\text{Prop} \to \{\text{true}, \text{false}\}$ denotes the set of all function from Prop to $\{\text{true}, \text{false}\}$

In Agda we define assignments and valuations as follows

```
Assignment : Set
Assignment = ℕ → 𝔹

⟦_⟧_ : Prop → Assignment → 𝔹
⟦ (Pvar n) ⟧ a = a n
⟦ ⊤ ⟧ _ = true
⟦ ⊥ ⟧ _ = false
⟦ (A ∧ B) ⟧ a = ⟦ A ⟧ a and ⟦ B ⟧ a
⟦ (A ∨ B) ⟧ a = ⟦ A ⟧ a or ⟦ B ⟧ a
⟦ (A ⟶ B) ⟧ a = ⟦ A ⟧ a cond ⟦ B ⟧ a
```

If we have a context Γ, we say that an assignment a satisfies Γ if $[\![\gamma]\!]^a = \text{true}$ for all γ in Γ

```
_satisfies_ : Assignment → Ctx → Set
a satisfies Γ = ∀ {γ} → γ ∈ Γ → ⟦ γ ⟧ a ≡ true
```

Finally, we say the Γ entails C semantically if $[\![C]\!]^a = \text{true}$ for all valuations a satisfying Γ, and we denote this as $\Gamma \vDash C$. In Agda we define it as

```
_⊨_ : Ctx → Prop → Set
Γ ⊨ C = ∀ {a} → a satisfies Γ → ⟦ C ⟧ a ≡ true
```

Before we can prove soundness, we need to prove a basic property of valuations.

Proposition 3.5.1 (Valuations are decidable): For any valuation a and any proposition A we have either

$$[\![\varphi]\!]^a = \text{true} \quad \text{or} \quad [\![\varphi]\!]^a = \text{false}$$

In Agda we state this as

```
val-dec : ∀ A a → Either (⟦ A ⟧ a ≡ true) (⟦ A ⟧ a ≡ false)
```

Proof: We proceed by induction on the complexity of the propositional formula φ

Base case 1 : $\varphi = P_n$

By the definition of valuations

$$[\![P_n]\!]^a = a(P_n),$$

and so we can determine the valuation by simply evaluating $a(P_n)$ and considering the result.

```
val-dec (Pvar n) a with a n
...                | true  = left  refl
...                | false = right refl
```

Base case 2 and 3 : $\varphi = \top$ and $\varphi = \bot$

These cases are trivial.

Inductive case 4 : $\varphi = A \to B$

By the inductive hypothesis, $[\![A]\!]^a$ and $[\![B]\!]^b$ are both decidable, so we can consider their valuations on a case-by-case basis.

For this reason, we can simply construct a truth-table to determine the valuation of $[\![A \to B]\!]^a$

$[\![A]\!]^a$	$[\![B]\!]^a$	$[\![A \to B]\!]^a$
true	true	true
true	false	false
false	true	true
false	false	true

This is essentially what we're doing in Agda too

```
val-dec (A ⟶ B) a with val-dec A a | val-dec B a
...                    | left h     | left  t = left  (cong₂ _cond_ h t)
...                    | left h     | right t = right (cong₂ _cond_ h t)
...                    | right h    | left  t = left  (cong₂ _cond_ h t)
...                    | right h    | right t = left  (cong₂ _cond_ h t)
```

Base case 5 and 6 : $\varphi = A \wedge B$ and $\varphi = A \vee B$

These cases are handled analogous to case 4.

□

Proving soundness is relatively straightforward, but first we need to state this simple lemma.

Lemma 3.5.1:

If x is false, then x is not true.

```
false-not-true : ∀ {x} → x ≡ false → x ≢ true
```

Theorem 3.5.1 (Soundness): If $\Gamma \vdash C$ then $\Gamma \vDash C$.

In Agda we state this as

```
soundness : ∀ {Γ} {C} → Γ ⊢ C → Γ ⊨ C
```

Proof:

We have to show that given a proof tree $\mathcal{D}$ of the sequent $\Gamma \vdash C$, and some arbitrary assignment a such that a satisfies Γ, we can prove that $[\![C]\!]^{a} = \text{true}$.

We will proceed by structural induction on the proof tree.

Base case 1 : $\mathcal{D} = \dfrac{P_n \in \Gamma}{\Gamma \vdash P_n}\,\text{id}$

a satisfies $C = P_n$ since a satisfies Γ and P_n is in Γ.

```
soundness (id Pₙ∈Γ) a-sat-Γ = a-sat-Γ Pₙ∈Γ
```

Base case 2 : $\mathcal{D} = \dfrac{\bot \in \Gamma}{\Gamma \vdash C}\,\bot_{\text{L}_2}$

This is a contradiction. Suppose a satisfies Γ, then, in particular $[\![\bot]\!]^{a} = \text{true}$, but we know by the definition of valuations that $[\![\bot]\!]^{a} = \text{false}$. There cannot be any a that satisfy Γ.

```
soundness (⊥L ⊥∈Γ) a-sat-Γ = bot-elim (false-not-true refl (a-sat-Γ ⊥∈Γ))
```

Inductive case 3 : $\mathcal{D} = \dfrac{\mathcal{D}_1 \quad \mathcal{D}_2}{\Gamma \vdash A \quad \Gamma \vdash B}{\scriptstyle \wedge_{\mathrm{R}}}$

By the inductive hypotheses on $\mathcal{D}_1$ and $\mathcal{D}_2$ we have

$$\Gamma \vDash A \quad \text{and} \quad \Gamma \vDash B.$$

So $[\![A]\!]^a = [\![B]\!]^a = \text{true}$, and so, by the definition of valuations, $[\![A \wedge B]\!] = \text{true}$.

```
soundness {Γ} {A ∧ B} (∧R 𝒟₁ 𝒟₂) a-sat-Γ = cong₂ _and_ (Γ⊨A a-sat-Γ) (Γ⊨B a-sat-Γ)
  where
  Γ⊨A : Γ ⊨ A
  Γ⊨A = soundness 𝒟₁

  Γ⊨B : Γ ⊨ B
  Γ⊨B = soundness 𝒟₂
```

Inductive case 4 : $\mathcal{D} = \dfrac{\mathcal{D}_1 \quad \mathcal{D}_2}{A \to B \in \Gamma \quad \Gamma \vdash A \quad \Gamma, B \vdash C}{\scriptstyle \to_{\mathrm{L}}}$

By the inductive hypothesis on $\mathcal{D}_1$ and $\mathcal{D}_2$ we have

$$\Gamma \vDash A \quad \text{and} \quad \Gamma, B \vDash C.$$

Since $A \to B \in \Gamma$ we also know that $[\![A \to B]\!]^a = \text{true}$. This gives us, by the definition of valuations, either

$$[\![A]\!]^a = \text{false} \quad \text{or} \quad [\![A]\!]^a = [\![B]\!]^a = \text{true}.$$

If $[\![A]\!]^a = \text{false}$ we have a contradiction as $\Gamma \vDash A$ implies $[\![A]\!]^a = \text{true}$, so it must be the case that $[\![A]\!]^a = [\![B]\!]^a = \text{true}$.

In this case, a satisfies not only Γ, but Γ, B, and since $\Gamma, B \vDash C$ we have $[\![C]\!]^a = \text{true}$, which is what we wanted to show.

```
soundness {Γ} {C} (→L {_} {A} {B} A→B∈Γ 𝒟₁ 𝒟₂) {a} a-sat-Γ = C-t
  where
  -- Γ ⊨ A by inductuve hypothesis.
  Γ⊨A : Γ ⊨ A
  Γ⊨A = soundness 𝒟₁

  -- Γ, B ⊨ C by inductuve hypothesis.
  ΓB⊨C : Γ , B ⊨ C
  ΓB⊨C = soundness 𝒟₂

  -- A valuates to true since Γ ⊨ A.
  A-t : ⟦ A ⟧ a ≡ true
  A-t = Γ⊨A a-sat-Γ

  -- A → B valuates to true since A → B ∈ Γ.
  A→B-t : ⟦ A ⟶ B ⟧ a ≡ true
  A→B-t = a-sat-Γ A→B∈Γ

  -- B valuates to true since A and A → B valuates to true.
  B-t : ⟦ B ⟧ a ≡ true
  B-t with val-dec B a
  ... | left A-t = A-t
  ... | right B-f = bot-elim (false-not-true (cong₂ _cond_ A-t B-f) A→B-t)

  -- Therefore a satisfies Γ, B.
  a-sat-ΓB : a satisfies (Γ , B)
  a-sat-ΓB head = B-t
  a-sat-ΓB (tail h) = a-sat-Γ h

  -- Finally, C valuates to true.
  C-t : ⟦ C ⟧ a ≡ true
  C-t = ΓB⊨C a-sat-ΓB
```

All other cases are handled analogously using the inductive hypothesis on sub-terms.

□

4. Proof searching

Here we develop the restricted sequent calculus that will be used to implement the search procedure, starting with an exposition that introduces the problem of termination in a search procedure. The inference rules themselves are introduced in Section 4.1.

The methodology of proving. The way we prove a statement in sequent calculus is by building a proof tree using the inference rules. The "root" of the tree is always the sequent we are trying to prove, and the leaves are always the inference rules id, $\top_{\mathrm{R}}$ or $\bot_{\mathrm{L}_2}$, i.e. those inference rules which have no further premises that need proving. For instance, consider the example proof tree of $P_1 \wedge P_2 \vdash P_2 \wedge P_1$ on page 11.

$$\dfrac{\dfrac{\dfrac{}{P_1 \wedge P_2, P_2 \vdash P_2}\,\mathrm{id}}{P_1 \wedge P_2 \vdash P_2}\,\wedge_{\mathrm{L}_2} \qquad \dfrac{\dfrac{}{P_1 \wedge P_2, P_1 \vdash P_1}\,\mathrm{id}}{P_1 \wedge P_2 \vdash P_1}\,\wedge_{\mathrm{L}_1}}{P_1 \wedge P_2 \vdash P_2 \wedge P_1}\,\wedge_{\mathrm{R}}$$

Once we have the proof tree, it might make sense to us, and we can verify that this proof tree is correct, but how do you come up with one? It is almost always easiest to find a proof tree starting from the bottom working upwards. In the example above we would start off like this:

$$\frac{?}{P_1 \wedge P_2 \vdash P_2 \wedge P_1}\,?$$

Now we look at Figure 1 and see if there are any inference rules that are applicable. We see that $\wedge_{\mathrm{R}}$, $\wedge_{\mathrm{L}_1}$, $\wedge_{\mathrm{L}_2}$ are all applicable. Now we have to make a choice as to which one of these we try – sometimes intuition helps us pick, sometimes trial and error is the only way. In any case, suppose we chose to apply $\wedge_{\mathrm{R}}$, now we end up with this:

$$\frac{\dfrac{?}{P_1 \wedge P_2 \vdash P_2}\,? \qquad \dfrac{?}{P_1 \wedge P_2 \vdash P_2}\,?}{P_1 \wedge P_2 \vdash P_2 \wedge P_1}\,\wedge_{\mathrm{R}}$$

Now we re-do this process for the two new "holes" in our proof tree, considering which inference rules are applicable, trying one of them, and (hopefully), eventually finding complete proof tree. Our proof searching procedure will operate on the same principle. However, it is not quite that simple as we can end up in infinite loops if we are not careful. Consider this attempt at proving commutativity that ends up in an infinite loop

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\vdots}{P_1 \wedge P_2, P_1, P_1, P_1, P_1 \vdash P_2 \wedge P_1}\,\wedge_{\mathrm{L}_1}}{P_1 \wedge P_2, P_1, P_1, P_1 \vdash P_2 \wedge P_1}\,\wedge_{\mathrm{L}_1}}{P_1 \wedge P_2, P_1, P_1 \vdash P_2 \wedge P_1}\,\wedge_{\mathrm{L}_1}}{P_1 \wedge P_2, P_1 \vdash P_2 \wedge P_1}\,\wedge_{\mathrm{L}_1}}{P_1 \wedge P_2 \vdash P_2 \wedge P_1}\,\wedge_{\mathrm{L}_1}$$

Figure 4: A search that goes on forever

How do we fix this? A first instinct might be to implement some sort of cycle-detection algorithm, but that can be difficult for two reasons

1. There might be cycles that are difficult to detect.
2. It might be very difficult to convince Agda that our search procedure with cycle detection does terminate.

The approach we take will instead be to restrict our inference rules to make these pathological cases impossible, and proving rigorously that this is the case. In the end we will arrive at a "contraction-free" sequent calculus along the lines of R. Dyckhoff [1]. Our search procedure will also follow a very specific order of applying inference rules, and we want the inference rules themselves to impose this order: For any given sequent there should be at most one or two inference rules that are applicable, i.e. at most one or two "choices" for the search procedure to consider. In order to achieve this restrictiveness we endow our sequents with a "cursor" and a "mode."

Definition 4.1 (Sequent with a cursor and a mode): Let $\Gamma \mid \Delta \vdash C \bullet m$ denote a sequent with a cursor and a mode, where Γ and Δ are contexts, C is a proposition, and m is one of the symbols R or S denoting reduce-mode and search-mode respectively. We may write

$$\Gamma, x \mid y, \Delta \vdash C \bullet m$$

to denote a sequent where x is the proposition just to the left of the cursor, and y is the proposition just to the right.

In Agda we define sequents with cursors as follows

```
record SequentWithCursorAndMode : Set₂ where
  constructor _|_⊢_•_
  field
    leftOfCursor : Ctx
    rightOfCursor : Ctx
    succedent    : Prop
    mode         : Mode
```

We want a way to build lists in "reverse order", because in the sequent $\Gamma, x \mid y, \Delta \vdash C \bullet m$ we have y as the head of the list y, Δ. We define this as follows

```
_,,_ : Prop → Ctx → Ctx
x ,, xs = xs , x
```

and so the above sequent is written as `Γ , x | y ,, Δ ⊢ C • m` in Agda.

4.1. The restricted sequent calculus LJf.

The overall strategy of our search procedure will be to simplify all of the propositions in the antecedents using left rules, starting from the right and moving left. When all of the propositions in the antecedents have been simplified as far as possible, and the cursor is all the way to the left, we will try to simplify the succedent using a right rule, and then "rewind" the cursor all the way to the right and restart our search. There will overall be three types of rules

$$\frac{\Gamma, x' \mid \Delta \vdash C \bullet m}{\Gamma, x \mid \Delta \vdash C \bullet m}\ \text{left rule} \qquad \frac{\Gamma \mid \emptyset \vdash C' \bullet m}{\emptyset \mid \Gamma \vdash C \bullet m}\ \text{right rule} \qquad \frac{\Gamma \mid x, \Delta \vdash C \bullet m}{\Gamma, x \mid \Delta \vdash C \bullet m}\ \text{move rule}$$

For all of the inference rules, the premises have to be strictly "smaller" than the conclusion according to some measure. Unfortunately, this measure cannot simply be the structural complexity of the sequent, and we will not define the specific measure until Section 5.

Now we use Figure 1 as a basis and restrict those rules one by one. The mode m exists in order to handle a specific scenario relating to propositions of the form $P_n \to B$ in the context, and will be discussed last. For all other inference rules, the mode will always be R, and there is no need to consider the mode until the last few inference rules.

Propositional variables.

If we encounter a propositional variable in the context, i.e. we find ourselves in this situation

$$\Gamma, P_n \mid \Delta \vdash C,$$

then we might check if $C = P_n$, at which point we've found our proof tree. If, on the other hand, $C \neq P_n$ then we have to go on with our search. This gives us the following two rules for propositional variables

$$\frac{}{\Gamma, P_n \mid \Delta \vdash P_n \bullet R}\,\mathrm{id} \qquad \frac{\Gamma \mid P_n, \Delta \vdash C \bullet R}{\Gamma, P_n \mid \Delta \vdash C \bullet R}\,\mathrm{id}\langle$$

The id rule simply ends our search with a proof tree and the id$\langle$ rule makes progress by moving the cursor further along to the left, and so the premises are strictly "smaller" than the conclusions in both cases.

Truth.

If we encounter $\top$ in the context, there's is nothing we can do with it, and we make progress by simply removing it from the context. If we encounter $\top$ as a succedent, then we have immediately found a proof tree. Note that we only consider the succedent *after* we have simplified all the propositions in the context.

$$\frac{\Gamma \mid \Delta \vdash C \bullet R}{\Gamma, \top \mid \Delta \vdash C \bullet R}\,\top_{\mathrm{L}} \qquad \frac{}{\emptyset \mid \Gamma \vdash \top \bullet R}\,\top_{\mathrm{R}}$$

We can see that the sequent weight of the premise in $\top L$ is strictly "smaller" than the conclusion.

Falsehood. This is pretty much the same rule as in Figure 1.

$$\frac{}{\Gamma, \bot \mid \Delta \vdash C \bullet R}\,\bot_{\mathrm{L}_2}$$

Conjunction.

For LJf there is only one left rule for conjunction instead of two

$$\frac{\Gamma, A, B \mid \Delta \vdash C \bullet R}{\Gamma, A \wedge B \mid \Delta \vdash C \bullet R}\,\wedge\mathrm{L}$$

The reasoning behind this restricted version is that with the original $\wedge_{\mathrm{L}_1}$ and $\wedge_{\mathrm{L}_2}$ rules it is possible to end up in an infinite loop as seen in Figure 4 by just repeatedly applying this rule. In this restricted, version that is not possible because it removes the assumption $A \wedge B$ from the context. This removal is not too restrictive, because assuming $A \wedge B$ and assuming A, B are logically equivalent, and so we don't lose any information. We also note that this makes the sequent in the premise strictly "smaller" in some sense.

The right rule for conjunction is nothing special.

$$\frac{\Gamma \mid \emptyset \vdash A \bullet R \qquad \Gamma \mid \emptyset \vdash B \bullet R}{\emptyset \mid \Gamma \vdash A \wedge B \bullet R}\,\wedge_{\mathrm{R}}$$

Disjunction.

There is nothing very special about the left rule for disjunction

$$\frac{\Gamma, A \mid \Delta \vdash C \bullet R \quad \Gamma, B \mid \Delta \vdash C \bullet R}{\Gamma, A \vee B \mid \Gamma \vdash C \bullet R} \vee_{\mathrm{L}}$$

nor with the two right rules

$$\frac{\Gamma \mid \emptyset \vdash A \bullet R}{\emptyset \mid \Gamma \vdash A \vee B \bullet R} \vee_{\mathrm{R}_1} \qquad \frac{\Gamma \mid \emptyset \vdash B \bullet R}{\emptyset \mid \Gamma \vdash A \vee B \bullet R} \vee_{\mathrm{R}_2}$$

Again, the sequents in the premises are strictly "smaller" than the conclusions.

There is one point to be made about branching: Sow far we've seen that the rules $\wedge_{\mathrm{R}}$ and $\vee_{\mathrm{L}}$ contain two premises each, meaning that our search procedure will have to branch off at these points. This branching is purely mechanical; If I want to prove for instance the sequent $\Gamma \vdash A \wedge B$, then i *know* that i will *always* have to prove $\Gamma \vdash A$ and $\Gamma \vdash B$ first. The $\vee_{\mathrm{R}_1}$ and $\vee_{\mathrm{R}_2}$ also induce branching in the procedure, but of another nature: If I want to prove $\Gamma \vdash A \vee B$ then I will have to prove either $\Gamma \vdash A$ or $\Gamma \vdash B$, but not necessarily both. This branching then represents some fundamental uncertainty; I only need to prove one of the sub-sequents but I don't know which one, so worst case, I'll have to try proving both.

Conditional

The right rule for conditional is straightforward

$$\frac{\Gamma, A \mid \emptyset \vdash B \bullet R}{\emptyset \mid \Gamma \vdash A \to B \bullet R} \to_{\mathrm{R}}$$

The left rule for conditionals poses a problem however. Consider the left-conditional rule from Figure 1

$$\frac{\Gamma, A \to B \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C} \to_{\mathrm{L}}$$

The problem is that if $A = C$, then the first premise is precisely the conclusion, i.e. we don't make any progress applying the rule like this, and we might end up in an infinite loop similar to Figure 4. The solution is to have six (!!) left-conditional rules specialized depending on the value of A.

Left-conditional with truth

If the conditional is $\top \to B$ then we may always assume B.

$$\frac{\Gamma, B \mid \Delta \vdash C \bullet R}{\Gamma, \top \to B \mid \Delta \vdash C \bullet R} \top \to \mathrm{L}$$

Left-conditional with falsehood

The conditional $\bot \to B$ is logically equivalent to $\top$, (assuming $\bot$, it follows that B for any B) and, just like $\top$, this proposition isn't useful, assuming it doesn't give us anything new to work with, leaving us with this rule

$$\frac{\Gamma \mid \Delta \vdash C \bullet R}{\Gamma, \bot \to B \mid \Delta \vdash C \bullet R} \bot \to \mathrm{L}$$

Left-conditional with conjunction

If the conditional is $(A_1 \wedge A_2) \to B$ then we simplify it with the logically equivalent $A_1 \to (A_2 \to B)$

$$\frac{\Gamma, A_1 \to (A_2 \to B) \mid \Delta \vdash C \bullet R}{\Gamma, (A_1 \wedge A_2) \to B \mid \Delta \vdash C \bullet R} \wedge \to \mathrm{L}$$

Here it is not very clear that the premise would be strictly "smaller" than the conclusion, however, in Section 5 we will see that it is possible to define a measure where this is the case. The intuition is that the proposition $A_1 \to (A_2 \to B)$ is "smaller" than $(A_1 \wedge A_2) \to B$ in the sense that the premise is smaller, and therefore we'll eventually end up with all conditionals in the context having the premises P_n, $\top$ or $\bot$.

Left-conditional with disjunction

If the conditional is $(A_1 \vee A_2) \to B$ then we know that $A_1 \to B$ and $A_2 \to B$, which is how we make progress in our search.

$$\frac{\Gamma, A_1 \to B, A_2 \to B \mid \Delta \vdash C \bullet R}{\Gamma, (A_1 \vee A_2) \to B \mid \Delta \vdash C \bullet R} \vee \to \mathrm{L}$$

Here it isn't obvious that the premise would be "smaller" either, but the intuition is that although there are more propositions in the premise, they "replaced" one large proposition $(A_1 \vee A_2) \to B$, and so we've made progress.

Left-conditional with conditional

Here the conditional is $(A_1 \to A_2) \to B$. If we came across this in LJ we might proceed with the following proof tree which we will use as a basis for finding the restricted version.

$$\frac{\dfrac{\dfrac{\vdots}{\Gamma, (A_1 \to A_2) \to B, A_1 \vdash A_2}}{\Gamma, (A_1 \to A_2) \to B \vdash A_1 \to A_2} \to_{\mathrm{R}} \quad \dfrac{\vdots}{\Gamma, (A_1 \to A_2) \to B, B \vdash C}}{\Gamma, (A_1 \to A_2) \to B \vdash C} \to_{\mathrm{L}}$$

The sequent in the second premise $\Gamma, (A_1 \to A_2) \to B, B \vdash C$ can have the assumption $(A_1 \to A_2) \to B$ removed without losing any information, because we already have B, and, assuming B we can conclude $X \to B$ for any X.

Similarly, the sequent in the first premise, $\Gamma, (A_1 \to A_2) \to B, A_1 \vdash A_2$ can be simplified; since we have A_1 in the context, the conditional $(A_1 \to A_2) \to B$ can be replaced with $A_2 \to B$ without losing any information.

These simplifications leave us with the following rule, which is a rule from LJT.

$$\frac{\Gamma, A_2 \to B, A_1 \vdash A_2 \quad \Gamma, B \mid \Delta \vdash C}{\Gamma, (A_1 \to A_2) \to B \vdash C} \to\!\to\mathrm{L}$$

By adding cursors and modes where appropriate, we arrive at the following inference rule for LJf.

$$\frac{\Gamma, A_2 \to B, A_1 \mid \Delta \vdash A_2 \bullet R \quad \Gamma, B \mid \Delta \vdash C \bullet R}{\Gamma, (A_1 \to A_2) \to B \mid \Delta \vdash C \bullet R} \to\!\to\mathrm{L}$$

Here the context of both premises is smaller – that is, if we use the same reasoning as for the left-conditional with disjunction. However, the first premise has A_2 as the succedent, whereas the succedent of the conclusions is C, and A_2 can be "larger" than C. Recall also that the $\to_{\mathrm{R}}$ rule reduces the succedent, but makes the context larger. This means that a well-founded measure cannot be a simple lexicographical ordering of

- the size of the context
- the size of the succedent

due to $\to\!\to$L and $\to_{\mathrm{R}}$.

Left-conditional with propositional variables

This is where we make use of the mode of the sequent.

If the conditional is $P_n \to B$ then there are two possible courses of action. If P_n is in the context, then we may of course assume B, giving us this rule

$$\frac{P_n \in \Gamma \cup \Delta \quad \Gamma, B \mid \Delta \vdash C \bullet R}{\Gamma, P_n \to B \mid \Delta \vdash C \bullet R} \mathrm{P}\to\!\to\mathrm{L}$$

If P_n isn't in the context, then we can't do anything with the $P_n \to B$ right now, however, once we've simplified all propositions then $P_n \to B$ might become applicable, so we don't discard it. This leaves us with a second inference rule for left-conditional with propositional variables.

$$\frac{\Gamma \mid P_n \to B, \Delta \vdash C \bullet R}{\Gamma, P_n \to B \mid \Delta \vdash C \bullet R} \mathrm{P}\to\!\langle$$

At this point, we are almost done, but there is one problem with this case. Consider for instance if we want to prove the sequent

$$\top \to P_1, P_1 \to P_2 \vdash P_2.$$

In such a search we start with the cursor all the way to the right, and in reduce-mode

$$\dfrac{}{\top \to P_1, P_1 \to P_2 \mid \emptyset \vdash P_2 \bullet R}\,?$$

We see first if the P⟶L rule is applicable; it isn't since P_1 is not an element in the context. P⟶⟨ is the only other applicable rule, leaving us with

$$\dfrac{\dfrac{}{\top \to P_1 \mid P_1 \to P_2 \vdash P_2 \bullet R}\,?}{\top \to P_1, P_1 \to P_2 \mid \emptyset \vdash P_2 \bullet R}\,\mathrm{P}{\longrightarrow}\langle$$

The ⊤⟶L rule is applicable, leaving us with

$$\dfrac{\dfrac{\dfrac{}{P_1 \mid P_1 \to P_2 \vdash P_2 \bullet R}\,?}{\top \to P_1 \mid P_1 \to P_2 \vdash P_2 \bullet R}\,\top{\longrightarrow}\mathrm{L}}{\top \to P_1, P_1 \to P_2 \mid \emptyset \vdash P_2 \bullet R}\,\mathrm{P}{\longrightarrow}\langle$$

Finally, we apply the id⟨ rule as it is the only applicable one

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \mid P_1, P_1 \to P_2 \vdash P_2 \bullet R}\,?}{P_1 \mid P_1 \to P_2 \vdash P_2 \bullet R}\,\mathrm{id}\langle}{\top \to P_1 \mid P_1 \to P_2 \vdash P_2 \bullet R}\,\top{\longrightarrow}\mathrm{L}}{\top \to P_1, P_1 \to P_2 \mid \emptyset \vdash P_2 \bullet R}\,\mathrm{P}{\longrightarrow}\langle$$

Now there are no more applicable inference rules and the search fails to find a proof of the sequent. We can recognize that there is a solution, tough. Since P_1 is now in the context we should be able to reduce $P_1 \to P_2$ to P_2 and continue with the search. The reason the search fails is due to the order of $\top \to P_1, P_1 \to P_2$ in the context. Had the order been reversed the search would have succeeded.

The way we remedy this is to perform one final pass over the context before giving up with the search, we do this with the following inference rule

$$\dfrac{C \in \{P_n \mid n \in \mathbb{N}\} \cup \{\bot\} \qquad \Gamma \mid \emptyset \vdash C \bullet S}{\emptyset \mid \Gamma \vdash C \bullet R}\,\mathrm{init}$$

This simply rewinds the cursor, and continues in "search" mode whenever the succedent C cannot be reduced further. In order for the premise to be considered "smaller" than the conclusion we say that for two equivalent sequents s_1 and s_2 (modulo order of the context and position of the cursor), the sequent s_1 is smaller than s_2 if s_1 is in search mode and s_2 is in reduce mode.

All previous inference rules are only applicable when in "reduce" mode, but we want the P⟶L rule to be applicable in both modes, so we change it

$$\frac{P_n \in \Gamma \cup \Delta \qquad \Gamma, B \mid \Delta \vdash C \bullet R}{\Gamma, P_n \to B \mid \Delta \vdash C \bullet m}\,\mathrm{P}{\to}\mathrm{L}$$

Note that the premise is always in reduce mode, meaning that when P⟶L is applied in search mode, the search "resets" back to reduce mode.

The final inference rule is only applicable in search mode, and is analogous to id⟨, but applicable for all propositions.

$$\frac{\Gamma \mid A, \Delta \vdash C \bullet S}{\Gamma, A \mid \Delta \vdash C \bullet S}\,\mathrm{continue}$$

With these additions and changes in place, there is now a proof for the sequent $\top \to P_1, P_1 \to P_2 \vdash P_2$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\top \to P_1, P_1 \to P_2 \mid \emptyset \vdash P_2 \bullet R}\,\mathrm{P}{\to}\langle}{\top \to P_1 \mid P_1 \to P_2 \vdash P_2 \bullet R}\,\top{\to}\mathrm{L}}{P_1 \mid P_1 \to P_2 \vdash P_2 \bullet R}\,\mathrm{id}\langle}{\emptyset \mid P_1, P_1 \to P_2 \vdash P_2 \bullet R}\,\mathrm{init}}{P_2 \in \{P_n \mid n \in \mathbb{N}\} \cup \{\bot\} \qquad P_1 \to P_2, P_1 \mid \emptyset \vdash P_2 \bullet S}\,\mathrm{continue}}{P_1 \to P_2 \mid P_1 \vdash P_2 \bullet S}\,\mathrm{P}{\to}\mathrm{L}}{P_1 \in \{P_1 \to P_2\} \cup \{P_1\} \qquad P_2 \mid P_1 \vdash P_2 \bullet R}\,\mathrm{id}$$

All inference rules are summarized in Figure 5

$$\frac{}{\Gamma, P_n \mid \Delta \vdash P_n}\,\mathrm{id} \qquad \frac{\Gamma \mid P_n, \Delta \vdash C}{\Gamma, P_n \mid \Delta \vdash C}\,\mathrm{id}\langle$$

$$\frac{}{\emptyset \mid \Gamma \vdash \top}\,\top_{\mathrm{R}} \qquad \frac{\Gamma \mid \Delta \vdash C}{\Gamma, \top \mid \Delta \vdash C}\,\top_{\mathrm{L}} \qquad \frac{}{\Gamma, \bot \mid \Delta \vdash C}\,\bot_{\mathrm{L}_2}$$

$$\frac{\Gamma \mid \emptyset \vdash A \quad \Gamma \mid \emptyset \vdash B}{\emptyset \mid \Gamma \vdash A \wedge B}\,\wedge_{\mathrm{R}} \qquad \frac{\Gamma, A, B \mid \Delta \vdash C}{\Gamma, A \wedge B \mid \Delta \vdash C}\,\wedge\mathrm{L}$$

$$\frac{\Gamma \mid \emptyset \vdash A}{\emptyset \mid \Gamma \vdash A \vee B}\,\vee_{\mathrm{R}_1} \quad \frac{\Gamma \mid \emptyset \vdash B}{\emptyset \mid \Gamma \vdash A \vee B}\,\vee_{\mathrm{R}_2} \qquad \frac{\Gamma, A \mid \Delta \vdash C \quad \Gamma, B \mid \Delta \vdash C}{\Gamma, A \vee B \mid \Delta \vdash C}\,\vee_{\mathrm{L}}$$

$$\frac{\Gamma, A \mid \emptyset \vdash B}{\emptyset \mid \Gamma \vdash A \to B}\,\to_{\mathrm{R}} \quad \frac{\Gamma, B \mid \Delta \vdash C}{\Gamma, \top \to B \mid \Delta \vdash C}\,\top\to\mathrm{L} \qquad \frac{\Gamma \mid \Delta \vdash C}{\Gamma, \bot \to B \mid \Delta \vdash C}\,\bot\to\mathrm{L}$$

$$\frac{\Gamma, A_1 \to (A_2 \to B) \mid \Delta \vdash C}{\Gamma, (A_1 \wedge A_2) \to B \mid \Delta \vdash C}\,\wedge\to\mathrm{L} \qquad \frac{\Gamma, A_1 \to B, A_2 \to B \mid \Delta \vdash C}{\Gamma, (A_1 \vee A_2) \to B \mid \Delta \vdash C}\,\vee\to\mathrm{L}$$

$$\frac{\Gamma, A_2 \to B, A_1 \mid \Delta \vdash A_2 \quad \Gamma, B \mid \Delta \vdash C}{\Gamma, (A_1 \to A_2) \to B \mid \Delta \vdash C}\,\to\to\mathrm{L}$$

$$\frac{P_n \in \Gamma \cup \Delta \quad \Gamma, B \mid \Delta \vdash C \bullet R}{\Gamma, P_n \to B \mid \Delta \vdash \bullet\, m}\,\mathrm{P}\to\mathrm{L} \qquad \frac{\Gamma \mid P_n \to B, \Delta \vdash C}{\Gamma, P_n \to B \mid \Delta \vdash C}\,\mathrm{P}\to\langle$$

$$\frac{C \in \{P_n \mid n \in \mathbb{N}\} \cup \{\bot\} \quad \Gamma \mid \emptyset \vdash C \bullet S}{\emptyset \mid \Gamma \vdash C \bullet R}\,\mathrm{init} \qquad \frac{\Gamma \mid A, \Delta \vdash C \bullet S}{\Gamma, A \mid \Delta \vdash C \bullet S}\,\mathrm{continue}$$

Figure 5: All inference rules for the restricted sequent calculus LJf. For brevity, the mode specifier is omitted for those inference rules where all premises and the conclusion are in reduce mode.

These inference rules are defined in Agda as follows:

```
data _∈LJf : SequentWithCursorAndMode → Set₂ where
  id-f⟨ : ∀ {Γ Δ} {n} {C}
        → Γ | Pvar n ,, Δ ⊢ C • R ∈LJf
          ---------------------- id-f⟨
        → Γ , Pvar n | Δ ⊢ C • R ∈LJf

  id  : ∀ {Γ Δ} {n}
          ------------------------- id
      → Γ , Pvar n | Δ ⊢ Pvar n • R ∈LJf

  ⊤R  : ∀ {Γ}
          ------------- ⊤R
      → ∅ | Γ ⊢ ⊤ • R ∈LJf

  ⊤L  : ∀ {Γ Δ} {C}
      → Γ | Δ ⊢ C • R ∈LJf
        ---------------- ⊤L
      → Γ , ⊤ | Δ ⊢ C • R ∈LJf

  ⊥L  : ∀ {Γ Δ} {C}
        ----------------- ⊥L
      → Γ , ⊥ | Δ ⊢ C • R ∈LJf

  ∧R  : ∀ {Γ} {A B}
      → Γ | ∅ ⊢ A • R ∈LJf
      → Γ | ∅ ⊢ B • R ∈LJf
        ----------------- ∧R
      → ∅ | Γ ⊢ A ∧ B • R ∈LJf

  ∧L  : ∀ {Γ Δ} {A B C}
      → Γ , A , B | Δ ⊢ C • R ∈LJf
        --------------------- ∧L
      → Γ , A ∧ B | Δ ⊢ C • R ∈LJf

  ∨R₁ : ∀ {Γ} {A B : Prop}
      → Γ | ∅ ⊢ A • R ∈LJf
        ----------------- ∨R₁
      → ∅ | Γ ⊢ A ∨ B • R ∈LJf

  ∨R₂ : ∀ {Γ} {A B : Prop}
      → Γ | ∅ ⊢ B • R ∈LJf
        ----------------- ∨R₂
      → ∅ | Γ ⊢ A ∨ B • R ∈LJf

  ∨L  : ∀ {Γ Δ} {A B C : Prop}
      → Γ , A | Δ ⊢ C • R ∈LJf
      → Γ , B | Δ ⊢ C • R ∈LJf
        --------------------- ∨L
      → Γ , A ∨ B | Δ ⊢ C • R ∈LJf

  →R  : ∀ {Γ} {A B : Prop}
      → Γ , A | ∅ ⊢ B • R ∈LJf
        ------------------- →R
      → ∅ | Γ ⊢ A ⟶ B • R ∈LJf

  ⊤→L : ∀ {Γ Δ} {B C : Prop}
      → Γ , B | Δ ⊢ C • R ∈LJf
        ---------------------- ⊤→L
      → Γ , ⊤ ⟶ B | Δ ⊢ C • R ∈LJf

  ⊥→L : ∀ {Γ Δ} {B C : Prop}
      → Γ | Δ ⊢ C • R ∈LJf
        ---------------------- ⊥→L
      → Γ , ⊥ ⟶ B | Δ ⊢ C • R ∈LJf

  ∧→L : ∀ {Γ Δ} {A₁ A₂ B C : Prop}
      → Γ , A₁ ⟶ (A₂ ⟶ B) | Δ ⊢ C • R ∈LJf
        ---------------------------- ∧→L
      → Γ , (A₁ ∧ A₂) ⟶ B | Δ ⊢ C • R ∈LJf

  ∨→L : ∀ {Γ Δ} {A₁ A₂ B C : Prop}
      → Γ , A₁ ⟶ B , A₂ ⟶ B | Δ ⊢ C • R ∈LJf
        ------------------------------- ∨→L
      → Γ , (A₁ ∨ A₂) ⟶ B | Δ ⊢ C • R ∈LJf

  →→L : ∀ {Γ Δ} {A₁ A₂ B C : Prop}
      → Γ , A₂ ⟶ B , A₁ | Δ ⊢ A₂ • R ∈LJf
      → Γ , B | Δ   ⊢ C • R ∈LJf
        ---------------------------- →→L
      → Γ , (A₁ ⟶ A₂) ⟶ B | Δ ⊢ C • R ∈LJf

  P→L : ∀ {Γ Δ} {n} {B C : Prop} {m}
      → Either (Pvar n ∈ Γ) (Pvar n ∈ Δ)
      → Γ , B | Δ ⊢ C • R ∈LJf
        -------------------------- P→L
      → Γ , Pvar n ⟶ B | Δ ⊢ C • m ∈LJf

  P→-f⟨ : ∀ {Γ Δ} {n} {B C}
        → Γ | Pvar n ⟶ B ,, Δ ⊢ C • R ∈LJf
          ------------------------- P→-f⟨
        → Γ , Pvar n ⟶ B | Δ ⊢ C • R ∈LJf

initSearch : ∀ {Γ} {C}
           → isPvarOr⊥ C
           → Γ | ∅ ⊢ C • S ∈LJf
             -------------- init
           → ∅ | Γ ⊢ C • R ∈LJf

continueSearch : ∀ {Γ Δ} {A C}
               → Γ | A ,, Δ ⊢ C • S ∈LJf
                 ----------------- cont
               → Γ , A | Δ ⊢ C • S ∈LJf
```

Figure 6: The set LJf defined in Agda

4.2. The search procedure.

First we state two simple lemmas that we will make us of.

```
-- It is decidable whether or not two propositions are equal.
prop-dec-≡ : (A : Prop) → (B : Prop) → Either (A ≡ B) (A ≢ B)
```

```
-- It is decidable whether or not a proposition is in a context or not.
dec-∈ : (φ : Prop) → (Γ : Ctx) → Either (φ ∈ Γ) (φ ∉ Γ)
```

Proposition 4.2.1:

For any sequent with a cursor s, there is either a derivation for s in LJf, or there is no such derivation.

In Agda we state this as

```
derivationFor : SequentWithCursorAndMode → Set₂
derivationFor (Γ | Δ ⊢ C • m) = Γ | Δ ⊢ C • m ∈LJf

{-# TERMINATING #-}
isProvable : (s : SequentWithCursorAndMode)
           → Either (derivationFor s) (¬ derivationFor s)
```

(More on the {-# TERMINATING #-} pragma in the proof)

The fact that we return a proof that no derivation exists in the case that the search fails is important. It establishes the fact that the search procedure is exhaustive; the only reason for it failing to find a derivation in LJf is if there literally is no such derivation in LJf. This does not necessarily mean, however, that the search is complete (i.e. finds a proof for all semantically true statements), because it could be that the inference rules in LJf are not expressive enough.

Proof:

We will proceed by induction on the "size" of the sequent s according to Definition 5.2, which follows in the next section.

Unfortunately, Agda's automatic termination checker does not see that our induction is sound, and so we have to annotate the search procedure with the {-# TERMINATING #-} pragma, which instructs Agda to trust our judgment. In Section 5.1 we will see how to make the inductive reasoning explicit so that Agda accepts our search procedure without the pragma.

All inductive cases of this proof will follow the same prototype: Tasked with finding a proof tree of $s \in \text{LJf}$ we consider which inference rules in Figure 5 can be used to prove it. There will never be more than two applicable rules, suppose for the demonstration that $\mathcal{R}$ is the only applicable rule, and $s_1 \in \text{LJf}$ and $s_2 \in \text{LJf}$ are its premises. Next, we apply the inductive hypothesis and search for proof trees $\mathcal{D}_1$ and $\mathcal{D}_2$ for $s_1 \in \text{LJf}$ and $s_2 \in \text{LJf}$ respectively. If the sub-searches succeed then we have found a proof of $s \in \text{LJf}$

$$\frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ s_1 \in \text{LJf} & s_2 \in \text{LJf}\end{array}}{s \in \text{LJf}}\,\mathcal{R}$$

Suppose on the other hand that the first search fails, then we know that

$$\neg s_1 \in \text{LJf}$$

Since $\mathcal{R}$ was the only applicable rule, and since one of its sub-terms lead to a contradiction, we conclude that there can be no proof of s either.

Base case 1 : $s = \emptyset \mid \Gamma \vdash \top \bullet R$

This case is trivial.

```
isProvable (∅ | _ ⊢ ⊤ • R) = left ⊤R
```

Base case 2 : $s = \emptyset \mid \Gamma \vdash C \bullet S$

Here the cursor has reached the end in search mode, and there are no applicable inference rules, so there is no proof tree for s.

```
isProvable (∅ | _ ⊢ _ • S) = right λ()
```

Inductive case 3 and 4 : $s = \emptyset \mid \Gamma \vdash P_n \bullet R$ and $s = \emptyset \mid \Gamma \vdash \bot \bullet R$

Suppose the succedent is P_n

We apply the inductive hypothesis and search for a proof tree $\mathcal{D}_1$ of the sequent

$$\Gamma \mid \emptyset \vdash P_n \bullet S$$

(recall that a sequent in search mode is considered "smaller" than a sequent in reduce mode)

If we find a proof tree $\mathcal{D}_1$ then the search succeeds with the following proof tree

$$\frac{\begin{array}{c}\mathcal{D}_1 \\ \Gamma \mid \emptyset \vdash P_n \bullet S\end{array}}{\emptyset \mid \Gamma \vdash P_n \bullet R}\,\text{init}$$

Otherwise there are no applicable inference rules, and there is no proof tree of s.

```
isProvable (∅ | Γ ⊢ Pvar n • R)
   with isProvable (Γ | ∅ ⊢ Pvar n • S)
... | left  𝒟₁ = left (initSearch tt 𝒟₁)
... | right h  = right λ{ (initSearch _ 𝒟₁) → h 𝒟₁ }
```

When the succedent is $\bot$ the case is handled analogously.

Inductive case 5 : $s = \emptyset \mid \Gamma \vdash A \vee B \bullet R$

The proof of s has to start with $\vee_{R_1}$ or $\vee_{R_2}$ as there are no other applicable inference rules. As such, we apply the inductive hypothesis and search for proofs trees $\mathcal{D}_1$ and $\mathcal{D}_2$ of $\Gamma \mid \emptyset \vdash A$ and $\Gamma \mid \emptyset \vdash B$.

If the first sub-search succeeds, we have a proof of s with the following proof tree

$$\frac{\begin{array}{c}\mathcal{D}_1 \\ \Gamma \mid \emptyset \vdash A \bullet R\end{array}}{\emptyset \mid \Gamma \vdash A \vee B \bullet R}\,\vee_{R_1}$$

Similarly, if the second sub-search succeeds, we have a proof tree of s using the $\vee_{R_2}$ rule.

If bot sub-searches fail then there are no applicable inference rules, and there is no proof tree of s.

```
isProvable (∅ | Γ ⊢ (A ∨ B) • R)
   with isProvable (Γ | ∅ ⊢ A • R)
   | isProvable (Γ | ∅ ⊢ B • R)
... | (left 𝒟₁)     | _              = left (∨R₁ 𝒟₁)
... | _             | (left 𝒟₂)      = left (∨R₂ 𝒟₂)
... | (right ¬𝒟₁) | (right ¬𝒟₂) = right λ{ (∨R₁ 𝒟₁) → ¬𝒟₁ 𝒟₁
                                          ; (∨R₂ 𝒟₂) → ¬𝒟₂ 𝒟₂
                                          }
```

Inductive case 6 : $s = \Gamma, P_n \mid \Delta \vdash C \bullet R$

The applicable rules in this case are id and id$\langle$. If $C = P_n$ then our search succeeds with the following proof tree

$$\frac{}{\Gamma, P_n \mid \Delta \vdash P_n \bullet R}\,\mathrm{id}$$

If $C \neq P_n$ then we apply the inductive hypothesis and search for a proof tree of the sequent $\Gamma \mid P_n, \Delta \vdash C$.

If the sub-search succeeds with a proof tree $\mathcal{D}_1$ of $\Gamma \mid P_n, \Delta \vdash C \bullet R$ then we also have a proof of $\Gamma, P_n \mid \Delta \vdash C \bullet R$ with the following proof tree

$$\frac{\begin{array}{c}\mathcal{D}_1\\ \Gamma \mid P_n, \Delta \vdash C \bullet R\end{array}}{\Gamma, P_n \mid \Delta \vdash C \bullet R}\,\mathrm{id}\langle$$

If $C \neq P_n$ and if the sub-search failed, there is no proof tree of s.

```
isProvable (Γ , Pₙ@(Pvar n) | Δ ⊢ C • R)
   with prop-dec-≡ Pₙ C
... | left refl = left id
... | right Pₙ≢C with isProvable (Γ | Pₙ ,, Δ ⊢ C • R)
...                | left  x = left (id-f⟨ x)
...                | right x = right λ{ id → Pₙ≢C refl
                                      ; (id-f⟨ y) → x y
                                      }
```

Inductive case 7 : $s = \Gamma, (A_1 \to A_2) \to B \mid \Delta \vdash C \bullet R$

A proof tree of s has to start with the $\to\to$L rule as there are no other applicable inference rules. We apply the inductive hypothesis and search for proof trees $\mathcal{D}_1$ and $\mathcal{D}_2$ of $\Gamma, A_2 \to B, A_1 \mid \Delta \vdash A_1$ and $\Gamma, B \mid \Delta \vdash C$ respectively. If both sub-searches succeed then our search succeeds with the proof tree

$$\frac{\Gamma, A_2 \to B, A_1 \mid \Delta \vdash A_2 \quad (\mathcal{D}_1) \qquad \Gamma, B \mid \Delta \vdash C \quad (\mathcal{D}_2)}{\Gamma, (A_1 \to A_2) \to B \mid \Delta \vdash C} \to\to\mathrm{L}$$

If any of the searches fail, there can be no proof tree.

```
isProvable (Γ , (A₁ ⟶ A₂) ⟶ B ∣ Δ ⊢ C • R)
   with isProvable (Γ , A₂ ⟶ B , A₁ ∣ Δ ⊢ A₂ • R) ∣ isProvable (Γ , B ∣ Δ ⊢ C • R)
... ∣ left 𝒟₁     ∣ left 𝒟₂      = left (→→L 𝒟₁ 𝒟₂)
... ∣ right ¬𝒟₁ ∣ _              = right λ{ (→→L 𝒟₁ _) → ¬𝒟₁ 𝒟₁ }
... ∣ _           ∣ right ¬𝒟₂ = right λ{ (→→L _ 𝒟₂) → ¬𝒟₂ 𝒟₂ }
```

Inductive case 7 : $s = \Gamma, P_n \to B \mid \Delta \vdash C \bullet m$

Here a proof tree of s can start with either P⟶L, or P⟶⟨.

We consider first if P_n there is a proof h of P_n being an element in the context Γ or Δ, suppose this is the case, we then apply the inductive hypothesis, searching for a proof tree $\mathcal{D}_1$ of the sequent

$$\Gamma, B \mid \Delta \vdash C \bullet R.$$

If there is such a proof tree, then our search succeeds with the following proof tree of s

$$\frac{P_n \in \Gamma \cup \Delta \quad (h) \qquad \Gamma, B \mid \Delta \vdash C \bullet R \quad (\mathcal{D}_1)}{\Gamma, P_n \to B \mid \Delta \vdash C \bullet m} \mathrm{P}{\to}\mathrm{L}$$

If it is not the case that such a proof tree exists, either because $\mathcal{D}_1$ or h is a contradiction, we continue, applying the inductive hypothesis again, searching for a proof tree $\mathcal{D}_2$ of the sequent

$$\Gamma \mid P_n \to B, \Delta \vdash C \bullet m.$$

If there is such a proof tree then our search succeeds with the following proof trees of s depending on the mode m.

$$\frac{\Gamma \mid P_n \to B, \Delta \vdash C \bullet R}{\Gamma, P_n \to B \mid \Delta \vdash C \bullet R} \mathrm{P}{\to}\langle \qquad \frac{\Gamma \mid P_n \to B, \Delta \vdash C \bullet S}{\Gamma, P_n \to B \mid \Delta \vdash C \bullet S} \mathrm{continue}$$

We show only the Agda case for $m = R$. The case for $m = S$ is handled analogously except for the final application of P→L being replaced by an application of continue

```
isProvable (Γ , Pₙ@(Pvar n) ⟶ B | Δ ⊢ C • R)
   with isProvable (Γ , B | Δ ⊢ C • R)
   | isProvable (Γ | Pₙ ⟶ B ,, Δ ⊢ C • R)
   | dec-∈ Pₙ Γ
   | dec-∈ Pₙ Δ
... | left 𝒟₁    | _              | left ∈Γ   | _          = left (P→L (left ∈Γ) 𝒟₁)
... | left 𝒟₁    | _              | _         | left ∈Δ    = left (P→L (right ∈Δ) 𝒟₁)
... | _          | left 𝒟₂        | _         | _          = left (P→-f⟨ 𝒟₂)
... | right ¬𝒟₁ | right ¬𝒟₂      | _         | _          =
   right λ{ (P→L _ 𝒟₁) → ¬𝒟₁ 𝒟₁
          ; (P→-f⟨ 𝒟₂) → ¬𝒟₂ 𝒟₂
          }
... | _          | right ¬𝒟₂      | right ∉Γ | right ∉Δ =
   right λ{ (P→L (left  ∈Γ) _) → ∉Γ ∈Γ
          ; (P→L (right ∈Δ) _) → ∉Δ ∈Δ
          ; (P→-f⟨ 𝒟₂) → ¬𝒟₂ 𝒟₂
          }
```

All remaining cases are handled analogously.

□

5. Proving termination

Here we prove first that the inductive arguments in Proposition 4.2.1 are all sound by giving a measure according to which all premises of all inference rules in Figure 5 are strictly smaller than their respective conclusions.

In Section 5.1 we use this measure to prove termination to Agda, leaving us with a version of isProvable that does not need a {-# TERMINATING #-} pragma.

We use the measure defined by A. S. Troelstra and H. Schwichtenberg [10] to measure the "weight" of propositions.

Definition 5.1 (Propositional weight)**:** The propositional weight $w(A)$ of a proposition A is given recursively by

$$
\begin{aligned}
w(P_n) &= 2\\
w(\top) &= 2\\
w(\bot) &= 2\\
(A \wedge B) &= w(A)(1 + w(B))\\
(A \vee B) &= 1 + w(A) + w(B)\\
(A \to B) &= 1 + w(A)w(B).
\end{aligned}
$$

The propositional weight can be extended to a whole sequent with

$$w(\Gamma \mid \Delta \vdash C \bullet m) = \sum_{1 \le i \le k} w(\Gamma_i) + \sum_{1 \le i \le h} w(\Delta_i) + w(C)$$

where k and h are the length of Γ resp. Δ.

This measure appears rather arbitrary, but we can verify via routine and tedious calculation that for almost all inference rules in Figure 5, the premises are strictly smaller than the conclusions

according to this measure, with the exceptions being id⟨, init and continue. If we add one to the weight of all sequents in reduce mode, then the premise in init is now strictly smaller than the conclusion. Finally, we can take care of id⟨ and continue, if we define our measure lexicographically, where for two sequents s_1 and s_2 of equal weight, s_1 is considered "smaller" than s_2 if the cursor of s_1 is further to the left. This leaves us with our final measure

Definition 5.2 (Sequent size)**:** The size of the sequent $\Gamma \mid \Delta \vdash C \bullet m$ is given by the tuple

$$\Big(w(\Gamma \mid \Delta \vdash C \bullet m) + f(m), \quad k\Big)$$

where f maps $R \mapsto 0$ and $S \mapsto 1$, and k is the length of Γ.

We say that a sequent s_1 is "smaller" than a sequent s_2 if the sequent size of s_1 is smaller when ordered lexicographically.

We can now verify that all applications of the inductive hypothesis in the proof of Proposition 4.2.1 are sound according to Definition 5.2. In Agda this means that isProvable is guaranteed to terminate, although Agda cannot see it automatically.

5.1. Proving termination in Agda

An approximate description[4] of Agdas automatic termination checker is that it tries to detect sound uses of structural induction, (in programming terms: recursion on sub-terms of function arguments). When we want to make an inductive argument that is not along the lines of structural induction, we can make use of the Acc datatype from Agdas standard library, where Acc _<_ x means that x is "accessible" w.r.t. the relation _<_. There is a single constructor

```
acc : (∀ {y} → y < x → Acc _<_ y) → Acc _<_ x,
```

meaning that $\prec$ is accessible w.r.t x if for every y such that $y \prec x$, y is also accessible. This means in effect that there are no infinite decsending chains $\cdots \prec x_3 \prec x_2 \prec x_1$. The intended use-case for Acc is to modify isProvable to have the following signature

```
isProvable' : (s : SequentWithCursor)
            → Acc _<_ s
            → Either (derivationFor s) (¬ derivationFor s)
```

where _<_ is Definition 5.2 but in Agda.

To illustrate, the inductive case 5 in the proof of Proposition 4.2.1 is handled as follows (with new additions in the code highlighted)

```
isProvable' (∅ | Γ ⊢ (A ∨ B) • R) (acc rs)
  with isProvable' (Γ | ∅ ⊢ A • R) (rs 𝓗)
     | isProvable' (Γ | ∅ ⊢ B • R) (rs 𝓚)
... | (left 𝒟₁)     | _ = left (∨R₁ 𝒟₁)
... | _             | (left 𝒟₂) = left  (∨R₂ 𝒟₂)
... | (right ¬𝒟₁ | (right ¬𝒟₂) = right λ { (∨R₁ 𝒟₁) → ¬𝒟₁ 𝒟₁
                                         ; (∨R₂ 𝒟₂) → ¬𝒟₂ 𝒟₂
                                         }
```

[4] See A. Abel [11] for an in-depth description of the type of algorithm underpinning Agda's termination checker.

whre $\mathcal{H}$ and $\mathcal{K}$ are proofs of

$$\Gamma \mid \emptyset \vdash A \bullet R \prec \emptyset \mid \Gamma \vdash (A \vee B) \bullet R$$

and

$$\Gamma \mid \emptyset \vdash B \bullet R \prec \emptyset \mid \Gamma \vdash (A \vee B) \bullet R$$

respectively.

With this change applied to all cases, Agdas termination checker will see that our induction is sound. Finally, we prove that Definition 5.2 is well-founded in Agda, which is stated in terms of accessibility and means that every sequent is accessible w.r.t. $\prec$

```
<-wf : ∀ { s } → Acc _<_ s
```

Now we have the final, guaranteed-by-Agda-to-terminate, form of isProvable

```
isProvable : (s : SequentWithCursor)
           → Either (derivationFor s) (¬ derivationFor s)
isProvable s = isProvable' s (<-wf s)
```

6. Translating LJf to LJ

If our search procedure finds a proof tree in LJf then we may want a corresponding proof tree in LJ. We will prove that this correspondence exists in this section.

For this proof we will be working with inference rules in both LJ and LJf, and though some of the inference rules share the same names, context will make it clear which one is refereed to. In Agda, the inference rules in LJ will be prefixed with LJ..

We state first some lemmas that we will make us of in the proof

```
,-mono-r-⊆' : ∀ {Γ : Ctx} {x} → Γ ⊆ Γ , x    ++-mono-r-⊆ : ∀ (Γ Δ : Ctx) → Γ ⊆ Γ ++ Δ

lemma₁ : ∀ Γ Δ x → x ∈ Γ → x ∈ Γ ++ Δ        lemma₂ : ∀ {Γ} → Γ ++ ∅ ⊆ ∅ ++ Γ

lemma₃ : ∀ Γ Δ B X → Γ , B ++ Δ ⊆ (Γ , X ++ Δ) , B

lemma₄ : ∀ Γ Δ X → Γ ++ (Δ , X) ⊆ (Γ , X) ++ Δ
```

,

Proposition 6.1: If $\Gamma \mid \Delta \vdash C \in$ LJf then $\Gamma +\!\!+ \Delta \vdash C \in$ LJ.

In Agda we state this as

```
translate : ∀ {Γ Δ} {C} {m} → Γ | Δ ⊢ C • m ∈LJf → Γ ++ Δ ⊢ C
```

Proof:

We will proceed by induction on the depth of the proof tree $\mathcal{D}$ of $\Gamma \mid \Delta \vdash C$.

Base case 1 : $\mathcal{D} = \dfrac{}{\Gamma, P_n \mid \Delta \vdash P_n \in \mathrm{LJf}}\,\mathrm{id}$

In this case we need to find a proof tree for $\Gamma, P_n \mathbin{+\!\!+} \Delta \vdash P_n \in \mathrm{LJ}$. We can see that $P_n \in \Gamma, P_n$, and so by lemma₁ we have a proof h of $P_n \in \Gamma, P_n \mathbin{+\!\!+} \Delta$. Now we can construct the following proof tree for $\Gamma', P_n \mathbin{+\!\!+} \Delta \vdash P_n \in \mathrm{LJ}$

$$\dfrac{\begin{matrix}h\\ P_n \in \Gamma, P_n \mathbin{+\!\!+} \Delta\end{matrix}}{\Gamma, P_n \mathbin{+\!\!+} \Delta \vdash P_n}\,\mathrm{id}$$

In Agda we handle the case like this

```
translate {Γ , Pₙ} {Δ} id = LJ.id (lemma₁ (Γ , Pₙ) Δ Pₙ head)
```

Base case 2 : $\mathcal{D} = \dfrac{}{\Gamma, \bot \mid \Delta \vdash C}\,\bot_{\mathrm{L}_2}$

Handled analogously to case 1.

Base case 3 : $\mathcal{D} = \dfrac{}{\emptyset \mid \Gamma \vdash \top}\,\top_{\mathrm{R}}$

This case is trivial.

Inductive case 4 : $\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 & \mathcal{D}_2\\ \Gamma \mid \emptyset \vdash A & \Gamma \mid \emptyset \vdash B\end{matrix}}{\emptyset \mid \Gamma \vdash A \wedge B}\,\wedge_{\mathrm{R}}$

In this case we need to find a proof tree for $\emptyset \mathbin{+\!\!+} \Gamma \vdash A \wedge B$. By the inductive hypothesis we have proof trees $\mathcal{E}_1$ and $\mathcal{E}_2$ of

$$\Gamma \mathbin{+\!\!+} \emptyset \vdash A \quad \text{and} \quad \Gamma \mathbin{+\!\!+} \emptyset \vdash B$$

respectively. By lemma₂ and strong weakening we have proof trees $\mathcal{E}'_1$ and $\mathcal{E}'_2$ of

$$\emptyset \mathbin{+\!\!+} \Gamma \vdash A \quad \text{and} \quad \emptyset \mathbin{+\!\!+} \Gamma \vdash B$$

respectively. We now construct a proof tree for $\emptyset \mathbin{+\!\!+} \Gamma \vdash A \wedge B$

$$\dfrac{\begin{matrix}\mathcal{E}'_1 & \mathcal{E}'_2\\ \emptyset \mathbin{+\!\!+} \Gamma \vdash A & \emptyset \mathbin{+\!\!+} \Gamma \vdash B\end{matrix}}{\emptyset \mathbin{+\!\!+} \Gamma \vdash A \wedge B}\,\wedge_{\mathrm{R}}$$

In Agda the case is handled like this

```
translate (∧R 𝒟₁ 𝒟₂) =
  LJ.∧R
    (strong-weaken
      lemma₂
      (translate 𝒟₁)
    )
    (strong-weaken
      lemma₂
      (translate 𝒟₂)
    )
```

Inductive case 5 : $\mathcal{D} = \dfrac{\Gamma, B \mid \Delta \vdash C \qquad \Gamma, P_n \to B \mid \Delta \vdash C}{P_n \in \Gamma \cup \Delta}\,\mathrm{P}{\to}\mathrm{L}$

```
translate {Γ' , Pₙ ⟶ B} {Δ} (P→L (left h) 𝒟₁) =
  LJ.→L
    (lemma₁ (Γ' , Pₙ ⟶ B) Δ (Pₙ ⟶ B) head)
    (LJ.id (++-mono-r-⊆ (Γ' , Pₙ ⟶ B) Δ (tail h)))
    (strong-weaken
      (lemma₃ Γ' Δ B (Pₙ ⟶ B))
      (translate 𝒟₁)
    )
```

Inductive case 6 : $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \mid P_n \to B, \Delta \vdash C}}{\Gamma, P_n \to B \mid \Delta \vdash C}\,\mathrm{P}{\to}\langle$

In this case we need to find a proof tree for $\Gamma, P_n \to B +\!\!+ \Delta \vdash C$. By the inductive hypothesis on $\mathcal{D}_1$ we have a proof tree $\mathcal{E}_1$ of

$$\Gamma +\!\!+ P_n \to B, \Delta \vdash C$$

by strong weakening and lemma₄ this is the same as

$$\Gamma, P_n \to B +\!\!+ \Gamma \vdash C$$

```
translate (P→-f⟨ {Γ} {Δ} {n} {B} 𝒟₁) =
  strong-weaken
  (begin
    Γ ++ Pvar n ⟶ B ,, Δ  ≡⟨⟩
    Γ ++ (Δ , Pvar n ⟶ B) ⊆⟨ lemma₄ Γ Δ (Pvar n ⟶ B) ⟩
    (Γ , Pvar n ⟶ B) ++ Δ
    ∎)
  (translate 𝒟₁)
```

Inductive case 7 : $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma, A_2 \to B, A_1 \mid \Delta \vdash A_2} \qquad \overset{\mathcal{D}_2}{\Gamma, B \mid \Delta \vdash C}}{\Gamma, (A_1 \to A_2) \to B \mid \Delta \vdash C} \to{\to}\mathrm{L}$

In this case we need to find a proof tree for $\Gamma, (A_1 \to A_2) \to B +\!\!+ \Delta \vdash C$, which, by the definition of $+\!\!+$ is the same as $\underbrace{\Gamma +\!\!+ \Delta, (A_1 \to A_2) \to B}_{\Omega} \vdash C$.

By the inductive hypothesis on $\mathcal{D}_1$ we have a proof tree $\mathcal{E}_1$ of

$$\Gamma, A_2 \to B, A_1 + +\Delta \vdash A_2.$$

By strong weakening this gives us a proof tree $\mathcal{E}_1'$ of

$$\Omega, A_1, A_2 \to B \vdash A_2$$

By the inductive hypothesis on $\mathcal{D}_2$ we have a proof tree $\mathcal{E}_2$ of

$$\Gamma, B +\!\!+ \Delta \vdash C.$$

By strong weakening this gives us a proof tree $\mathcal{E}_2'$ of

$$\Omega, B \vdash C$$

By strong identity we have a proof tree $\mathcal{E}_3$ of $\Omega, A_1, A_2, A_1 \vdash A_2$.

By strong identity we have a proof $\mathcal{E}_4$ of $\Omega, A_1, A_2, B \vdash B$.

Now we may construct a proof of $\Gamma, (A_1 \to A_2) \to B +\!\!+ \Delta \vdash C$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{E}_3}{\Omega, A_1, A_2, A_1 \vdash A_2}}{\Omega, A_1, A_2 \vdash (A_1 \to A_2)} \to_{\mathrm{R}} \quad \dfrac{\mathcal{E}_4}{\Omega, A_1, A_2, B \vdash B}}{\Omega, A_1, A_2 \vdash B} \to_{\mathrm{L}}}{\Omega, A_1 \vdash A_2 \to B} \to_{\mathrm{R}} \quad \dfrac{\mathcal{E}_1'}{\Omega, A_1, A_2 \to B \vdash A_2}}{\dfrac{\Omega, A_1 \vdash A_2}{\Omega \vdash A_1 \to A_2}} \to_{\mathrm{R}} \quad \dfrac{\mathcal{E}_2'}{\Omega, B \vdash C}}{\Omega \vdash C} \to_{\mathrm{L}}$$ cut

```
translate (→→L {Γ} {Δ} {A₁} {A₂} {B} 𝒟₁ 𝒟₂) =
  LJ.→L
    head
    (LJ.→R
      (LJ.cut
        (LJ.→R
          (LJ.→L
            (tail (tail head))
            (LJ.→R (strong-id (tail head)))
            (strong-id head)
          )
        )
        (strong-weaken
          (λ{ head → tail head
            ; (tail head) → head
            ; (tail (tail h)) → tail (tail (tail h))
            }
          )
        (translate 𝒟₁))
      )
    )
    (strong-weaken
      (,-mono-⊆ ,-mono-r-⊆')
      (translate 𝒟₂)
    )
```

All other cases are handled analogously.

□

7. Conclusions and future work

We have developed a proof searching procedure for IPC in Agda, based on sequent calculus, and proven it to be correct. Here is a proof tree of disjunction being commutative as given by the search procedure and as a pen-and-paper proof

```
∨L (id-f⟨ (∨R₂ id)) (id-f⟨ (∨R₁ id))
```

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P_1 \mid \emptyset \vdash P_1 \bullet R}\,\mathrm{id}}{\emptyset \mid P_1 \vdash P_2 \vee P_1 \bullet R}\,\vee_{R_2}}{P_1 \mid \emptyset \vdash P_2 \vee P_1 \bullet R}\,\mathrm{id}\langle \qquad \dfrac{\dfrac{\dfrac{}{P_2 \mid \emptyset \vdash P_2 \bullet R}\,\mathrm{id}}{\emptyset \mid P_2 \vdash P_2 \vee P_1 \bullet R}\,\vee_{R_1}}{P_1 \mid \emptyset \vdash P_2 \vee P_1 \bullet R}\,\mathrm{id}\langle}{P_1 \vee P_2 \mid \emptyset \vdash P_2 \vee P_1 \bullet R}\,\vee_L$$

The proof tree in LJf can then be translated to a proof tree in LJ, giving us

```
∨L head (∨R₂ (id head)) (∨R₁ (id head))
```

$$\dfrac{\dfrac{}{P_1 \vee P_2 \in P_1 \vee P_2}\,\text{head} \qquad \dfrac{\dfrac{\dfrac{}{P_1 \in P_1 \vee P_2, P_1}\,\text{head}}{P_1 \vee P_2, P_1 \vdash P_1}\,\text{id}}{P_1 \vee P_2, P_1 \vdash P_2 \vee P_1}\,\vee_{R_2} \qquad \dfrac{\dfrac{\dfrac{}{P_2 \in P_1 \vee P_2, P_2}\,\text{head}}{P_1 \vee P_2, P_2 \vdash P_2}\,\text{id}}{P_1 \vee P_2, P_2 \vdash P_2 \vee P_1}\,\vee_{R_1}}{P_1 \vee P_2 \vdash P_2 \vee P_1}\,\vee_L$$

If the search procedure fails to find a proof tree for the sequent s it returns a function s → bot, i.e. a proof that no proof tree of s exists. Here is a proof that the law of excluded middle does not hold, as given by the proof searching procedure (after some manual cleanup)

```
λ { (∨R₁ (initSearch _ ()))
  ; (∨R₂ (→R (id-f⟨ (initSearch _ (continueSearch ((((((
  ; (∨R₂ (initSearch () _))
  ; (initSearch () _)
  }
```

The procedure can also handle more complicated problems, for instance, it can find a proof tree of a tripple-negated version of an unsatisfiable boolean formula in conjunctive normal form

$$\neg\neg\neg\Big((P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_3)$$
$$\wedge (\neg P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee P_2 \vee \neg P_3) \wedge (\neg P_1 \vee \neg P_2 \vee P_3) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3)\Big).$$

However, the proof tree is quite lengthy, with over 27300 inference rules. Here are the first seven lines of the proof tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\bot\to\bot, ((P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_3) \wedge (\neg P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee P_2 \vee \neg P_3) \wedge (\neg P_1 \vee \neg P_2 \vee P_3)) \mid \neg P_1 \vdash \bot \bullet R}\,\wedge L}{\bot\to\bot, ((P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_3) \wedge (\neg P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee P_2 \vee \neg P_3) \wedge (\neg P_1 \vee \neg P_2 \vee P_3)), \neg P_1 \mid \emptyset \vdash \bot \bullet R}\,\text{P}{\to}\langle}{\bot\to\bot, ((P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_3) \wedge (\neg P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee P_2 \vee \neg P_3) \wedge (\neg P_1 \vee \neg P_2 \vee P_3)), \neg P_1 \vee \neg P_2 \mid \emptyset \vdash \bot \bullet R}\,\vee_L}{\bot\to\bot, ((P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_3) \wedge (\neg P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee P_2 \vee \neg P_3) \wedge (\neg P_1 \vee \neg P_2 \vee P_3)), \neg P_1 \vee \neg P_2 \vee \neg P_3 \mid \emptyset \vdash \bot \bullet R}\,\vee_L}{\bot\to\bot, ((P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_3) \wedge (\neg P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee P_2 \vee \neg P_3) \wedge (\neg P_1 \vee \neg P_2 \vee P_3) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3)) \mid \emptyset \vdash \bot \bullet R}\,\wedge L \qquad \dfrac{}{\bot \mid \emptyset \vdash \bot \bullet R}\,\bot_{L_2}}{\neg\neg\Big((P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_3) \wedge (\neg P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee P_2 \vee \neg P_3) \wedge (\neg P_1 \vee \neg P_2 \vee P_3) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3)\Big) \mid \emptyset \vdash \bot \bullet R}\,{\to}{\to}L}{\emptyset \mid \emptyset \vdash \neg\neg\neg\Big((P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_3) \wedge (\neg P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee P_2 \vee \neg P_3) \wedge (\neg P_1 \vee \neg P_2 \vee P_3) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3)\Big) \bullet R}\,\neg_R$$

Future work

A natural direction for future research is to prove that the search procedure is complete in Agda. Since we have already established that the search procedure is correct it would be enough to prove this weaker version of completeness

```
completeness-irrefutable : ∀{Γ}{C} → Γ ⊨ C → ¬ ¬ (Γ ∣ ∅ ⊢ C ∈LJf)
```

The benefit of this approach is that we may employ classical reasoning for this proof using double negation translation.

Bibliography

[1] R. Dyckhoff, "Contraction-Free Sequent Calculi for Intuitionistic Logic," *The Journal of Symbolic Logic*, vol. 57, no. 3, pp. 795–807, 1992, Accessed: May 15, 2025. [Online]. Available: http://www.jstor.org/stable/2275431

[2] R. J. Simmons, "Structural focalization," *CoRR*, 2011, [Online]. Available: http://arxiv.org/abs/1109.6273

[3] G. Gentzen, "Untersuchungen über das logische Schließen. II," *Mathematische Zeitschrift*, vol. 39, no. 1, pp. 405–431, Dec. 1935, doi: 10.1007/BF01201363.

[4] F. Pfenning, "Lecture Notes on Sequent Calculus." [Online]. Available: https://www.cs.cmu.edu/~fp/courses/15317-f17/lectures/09-seqcalc.pdf

[5] E. Bastås, "proof-searching-for-ipc-in-agda." [Online]. Available: https://codeberg.org/emmabastas/proof-searching-for-ipc-in-agda

[6] P. Dybjer and E. Palmgren, "Intuitionistic Type Theory," *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2024.

[7] "Equality and equational reasoning." Aug. 2022. [Online]. Available: https://plfa.inf.ed.ac.uk/22.08/Equality/

[8] J. Carlström, *Logic*. Stockholm: Stockholms universitet, Matematiska insitutionen, 2013.

[9] J. von Plato, "The Development of Proof Theory," *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2018.

[10] A. S. Troelstra and H. Schwichtenberg, "Cut elimination with applications," in *Basic Proof Theory*, in Cambridge Tracts in Theoretical Computer Science., Cambridge University Press, 2000, p. 113.

[11] A. Abel, "foetus – Termination Checker for Simple Functional Programs," 1998, [Online]. Available: https://arxiv.org/abs/2407.06924