Master Parisien de Recherche en Informatique (MPRI) 2025–2026
Proof Assistants (PRFA)

Provability in natural deduction

`v1.5, 26 October`

Please hand in a zip file containing the three exercises called `ex1.v`, `ex2.v`, `ex3.v`, `ex4.v` and `ex5.v`, as well as a `README.txt` file explaining how to build and navigate your proofs. **The subject of your email should be "[MPRI PRFA] Project"** or you risk it being missed by us. Our emails: `yannick.forster@inria.fr` and `theo.winterhalter@inria.fr`.

Asking for help. You are encouraged to ask questions about the project on Discord.

You are allowed to discuss the project with your fellow students. You are *not* allowed to share code in any way: By copy-pasting, sending it to others, or looking at others' code.

Please start your `README.txt` file with a summary of who you discussed what aspect with. This includes both questions you ask fellow students *and* help you give to fellow students. An example for this is

- I asked X about whether strong induction is needed in exercise 5.
- I was asked by Y whether I defined the predicate in exercise 7 using `Fixpoint` or `Inductive`.

You are not allowed to use `ChatGPT` or similar tools.

README. Submit a `README.txt` file where you

- explain how to build the project,
- explain who you discussed the project with,
- explain your previous experience with Rocq or other proof assistants,
- state which exercises you did not solve and explain why you got stuck.

Building projects. Ensure that your project builds with Rocq `9.0`. You are allowed to use the Equations package. To help us read your project, please identify to which answer you reply to by using comments with questions numbers such as `(* 1.2.b *)`.

Comments. Please add comments to your Rocq file explaining design choices and difficulties you encountered when they do not fit in the README.

Evaluation. By solving exercises 1 to 3, you can obtain 14 out of 20 points.

Advice. Take a step back whenever you are stuck. Doing Rocq proofs can sometimes feel like a video game. If that happens, maybe you need to take a break to reflect on how you want to prove the thing. It might also help to do it on paper in those cases.

Changes. We will publish new versions of this PDF in case it becomes necessary, *i.e.* fixing typos or mistakes. We will update the version number and attach a changelog. You will of course be notified by email and on the Discord.

- `v1.5` Fixed typo in the proof of Lemma 1.

- `v1.4` Fixed typo in third item of Lemma 9. Fixed typo where we wrote `nd` instead of `ndm` just before Theorem 7.
- `v1.3` Notation in 1.1.b was incorrectly $\vdash$ when it should have been $\vdash_{\mathtt{c}}$.
- `v1.2` Fix minor typos.
- `v1.1`. Fix typo in the rules of classical natural deduction (we had written s, A instead of $s :: A$).
- `v1.0`. Initial version.

Deadline: 13 November 2025 at 18:00.

Late submissions are only accepted if you talk to us before November 6th.

1 Classical propositional logic

In this exercise, we will define a natural deduction system for classical propositional logic.

1.1 Classical natural deduction

Mathematically, the classical natural deduction system we consider has 4 rules (assumption, implication introduction, implication elimination, and proof by contradiction):

$$\frac{s \in A}{A \vdash_c s} \qquad \frac{s :: A \vdash_c t}{A \vdash_c s \to t} \qquad \frac{A \vdash_c s \to t \qquad A \vdash_c s}{A \vdash_c t} \qquad \frac{\neg s :: A \vdash_c \bot}{A \vdash_c s}$$

Start a file `ex1.v` with the following definitions and notations.

```
From Stdlib Require Import List.
Import ListNotations.

Inductive form : Type :=
| var (x : nat) | bot | imp (s t : form).

Print In.
Print incl.

Notation "s ~> t" := (imp s t) (at level 51, right associativity).
Notation neg s := (imp s bot).
Reserved Notation "A ⊢c s" (at level 70).
```

a. Define an inductive predicate `ndc : list form -> form -> Prop` capturing the rules from above. Declare the notation `A ⊢c s` for `ndc A s`.

b. Construct natural deduction proofs of the following statements:

1. `A ⊢c s ~> s`
2. `s :: A ⊢c neg (neg s)`
3. `[neg (neg bot)] ⊢c bot`. Can you do it without using proof by contradiction?
4. `A ⊢c (neg (neg s)) ~> s`

c. Prove weakening:

```
Fact Weakc A B s :
  A ⊢c s -> incl A B -> B ⊢c s.
```

d. Define a predicate `ground : form -> Prop` ensuring that no variables occur in a formula.

1.2 Model-based semantics

We are now going to build models of classical natural deduction which will help us deduce for instance consistency. In this exercise, we're going to use the following definition of model:

```
Definition Model := nat -> Prop.
```

a. Define a function `interp : model -> form -> Prop` that takes `M : Model` as argument to interpret variables. The definition we want should be of the following form:

$$\begin{aligned} [\![\bot]\!]_M &:= \bot \\ [\![s \to t]\!]_M &:= [\![s]\!]_M \Longrightarrow [\![t]\!]_M \\ [\![x]\!]_M &:= M(x) \end{aligned}$$

b. We now extend this definition to context as follows:

$$\begin{aligned} [\![[\,]]\!]_M &:= \top \\ [\![s :: A]\!]_M &:= [\![s]\!]_M \wedge [\![A]\!]_M \end{aligned}$$

Implement this as a function `ctx_interp : model -> list form -> Prop`.

c. We are now going to prove consistency, *classically*, *i.e.* by assuming double negation elimination `DNE : forall P, ~~P -> P`. To that end, we prove the following soundness lemma of classical natural deduction w.r.t. model-based semantics:

```
Lemma soundness M A (s : form) :
  (forall P, ~~P -> P) ->
  A ⊢c s ->
  ctx_interp M A ->
  interp M s.
```

d. Deduce consistency from soundness:

```
Lemma classical_consistency :
  (forall P, ~~P -> P) -> ~([] ⊢c bot).
```

e. Classical logic is in fact not necessary to derive consistency, and we can actually build a constructive proof of soundness for the model. Prove the following lemma, *without assuming classical axioms*:

```
Lemma constructive_soundness M A (s : form) :
  ndc A s -> ctx_interp M A -> ~~interp M s.
```

f. Deduce consistency:

```
Lemma constructive_consistency :
  ~ ([] ⊢c bot).
```

Note that classical natural deduction is actually complete for model-based semantics: If a formula holds in all models, it is provable. We do not prove this fact.

2 Minimal propositional logic

2.1 Minimal natural deduction

a. Minimal natural deduction can be defined by removing the rule for proofs by contradiction from natural deduction. Note that in particular, there is not even an explosion rule. Define it as a predicate `ndm : list form -> form -> Prop` with notation `A ⊢m s`.

b. Prove

```
Lemma Weakm A B s :
  A ⊢m s -> incl A B -> B ⊢m s.
```

c. Prove that minimally provable formulas are classically provable:

```
Lemma Implication A s :
  A ⊢m s -> A ⊢c s.
```

d. Define the Friedman translation `trans : form -> form -> form` such that `trans t s` replaces every occurrence of `bot` in `s` by `t` and `var x` by `(var x ~> t) ~> t`.

e. Prove

```
Lemma DNE_Friedman A s t :
  A ⊢m ((trans t s ~> t) ~> t) ~> (trans t s).
```

f. Prove

```
Lemma Friedman A s t :
  A ⊢c s -> map (trans t) A ⊢m trans t s.
```

g. Deduce that minimal and classical natural deduction derive the same ground formulas:

```
Lemma ground_truths s :
  ground s -> ([] ⊢m s <-> []⊢c s).
```

h. Deduce that minimal natural deduction is consistent if and only if classical natural deduction is consistent, in other words that one proves $\perp$ if and only if the other does.

i. From this we are able to deduce consistency of DNE in minimal logic. First we define the DNE formula:

```
Definition dne s := ((s ~> bot) ~> bot) ~> s.
```

Now prove the following lemma:

```
Lemma consistency_of_dne s :
  ~([] ⊢m dne s ~> bot).
```

2.2 World-based semantics

We are now going to explore a sound and complete semantics for minimal natural deduction: world-based semantics, sometimes also called Kripke semantics.

We again define a type of models, which is more complex than before. A model M is given by a type of worlds W_M, a binary relation on worlds $\leq_M$ and, for every world $w \in W_M$, a proposition $\bot_M(w)$ as well as an interpretation of variables into propositions $\mu_M(w, x)$ such that

- $\leq_M$ is reflexive and transitive,
- $w \leq_M w' \wedge \bot_M(w) \implies \bot_M(w')$,
- $w \leq_M w' \wedge \mu_M(w, x) \implies \mu_M(w', x)$.

a. Define a type `WModel` of world-based models as described above. Once this is done you can make use of the following notation.

```
Notation "w '≤(' M ) w'" := (M.(rel) w w') (at level 40, w' at next level).
```

b. Interpretation in the model is defined for every world by recursion on formulas:

$$
\begin{array}{rcl}
[\![\bot]\!]_M(w) & := & \bot_M(w) \\
[\![s \to t]\!]_M(w) & := & \forall w'.\ w \leq_M w' \wedge [\![s]\!]_M(w') \implies [\![t]\!]_M(w') \\
[\![x]\!]_M(w) & := & \mu_M(w, x)
\end{array}
$$

Define a function `winterp : forall (M : WModel), M.(world) -> form -> Prop` respecting the description above. Depending on how you defined `WModel` you may have to adapt the type above.

c. Extend interpretation to contexts, yielding

```
ctx_winterp : forall (M : WModel), M.(world) -> list form -> Prop
```

d. Show monotonicity of the interpretation:

```
Lemma monotonicity M s w w' :
  w ≤(M) w' -> winterp M w s -> winterp M w' s.
```

e. Extend monotonicity to contexts:

```
Lemma ctx_monotonicity M A w w' :
  w ≤(M) w' -> ctx_winterp M w A -> ctx_winterp M w' A.
```

f. We now establish soundness of minimal natural deduction w.r.t. world-based semantics.

```
Lemma wsoundness M A s :
  A ⊢m s -> forall w, ctx_winterp M w A -> winterp M w s.
```

g. We can establish consistency again by providing a model $\mathbb{1}$ with only one world $\star$ such that $\mu_{\mathbb{1}}(\star, x) := \top$ and $\bot_{\mathbb{1}}(\star) := \bot$. Define such a `consistency_model : WModel`.

h. Deduce consistency from it.

```
Lemma consistency :
  ~ ([] ⊢m bot).
```

i. More interesting models exist and allow us to derive more properties than mere consistency. For instance, a two-world model $\mathbb{2}$ containing 0 and 1 with $0 \leq_{\mathbb{2}} 1$ (but not the other way around) such that $\mu_{\mathbb{2}}(0, x) := \bot$ and $\mu_{\mathbb{2}}(1, x) := \top$ and $\bot_{\mathbb{2}}(w) := \bot$ will allow us to show independence of DNE. First, define this model `notdne_model : WModel`.

j. Use it to show independence of DNE.

```
Lemma dne_independent :
  ~(forall s, [] ⊢m dne s).
```

2.3 Completeness

We now show completeness of minimal natural deduction w.r.t. model-based semantics: If a formula holds in all world-based models, it is actually provable in natural deduction.

To that end, we are going to build a *syntactic model* $\mathbb{S}$. $W_{\mathbb{S}}$ is given by lists of formulas, $\leq_{\mathbb{S}}$ by inclusions (`incl`), while $\bot_{\mathbb{S}}(A)$ and $\mu_{\mathbb{S}}(A, s)$ are given by provability in minimal natural deduction judgments $A \vdash_m \bot$ and $A \vdash_m s$ respectively.

a. Define `syntactic_model : WModel` as described above.

b. Show that interpretation in this model coincides with provability.

```
Lemma correctness A s :
  winterp syntactic_model A s <-> A ⊢m s.
```

c. Deduce completeness.

```
Lemma completeness A s :
  (forall M w, ctx_winterp M w A -> winterp M w s) -> A ⊢m s.
```

3 Cut Elimination

"Cut" is a historical term for an application of the modus ponens rule deducing t from proofs of $s \to t$ and s for arbitrary formulas s. Cut-free proofs are restricted to implications $s_1 \to \dots s_n \to t \in A$.

Cut elimination shows that every natural deduction proof can be given in a cut-free way. It is fairly easy to show that there is no cut-free proof of double negation elimination.

For this exercise, we use mathematical notation. Your task is to formalise all notions in Rocq, and prove the lemmas and theorems. You will not have to invent mathematical arguments or intermediate lemmas unless explicitly stated. If not explicitly stated, the structure of the proof is given on paper without gaps.

Intuitively, a cut-free proof is either an implication introduction, or elimination of an implication which is an assumption, with cut-free proofs of the premise. We formalise this as predicates $A \vdash_{\mathrm{cf}} s$ and $A \vdash_{\mathrm{ae}} s$ mutually with the following rules (implication introduction, inclusion, implication elimination, assumption):

$$\frac{s :: A \vdash_{\mathrm{cf}} t}{A \vdash_{\mathrm{cf}} s \to t} \qquad \frac{A \vdash_{\mathrm{ae}} s}{A \vdash_{\mathrm{cf}} s} \qquad \frac{A \vdash_{\mathrm{ae}} s \to t \qquad A \vdash_{\mathrm{cf}} s}{A \vdash_{\mathrm{ae}} t} \qquad \frac{s \in A}{A \vdash_{\mathrm{ae}} s}$$

If you call the predicates `cf` and `ae` in Rocq, you can use the following command to generate a mutual induction principle:

```
Scheme cf_ind_mut := Induction for cf Sort Prop
    with ae_ind_mut := Induction for ae Sort Prop.
Combined Scheme cf_ae_ind from cf_ind_mut, ae_ind_mut.
Check cf_ae_ind.
```

Both predicates have the expected weakening property.
Lemma 1 (Weakening)**.** *The following properties hold for all $A \subseteq B$.*

- *If $A \vdash_{\mathrm{cf}} s$ then $B \vdash_{\mathrm{cf}} s$.*
- *If $A \vdash_{\mathrm{ae}} s$ then $B \vdash_{\mathrm{ae}} s$.*

Proof. By mutual induction on $A \vdash_{\mathrm{cf}} s$ and $A \vdash_{\mathrm{ae}} s$. The proof is very similar to the previous weakening proofs. □

We will now use cut-free proofs to build a cut-free syntactic model $\mathbb{C}$. $W_{\mathbb{C}}$ is again given by list of formulas related by inclusion ($A \leq_{\mathbb{C}} B := A \subseteq B$), and by taking $\bot_{\mathbb{C}}(A) := A \vdash_{\mathrm{cf}} \bot$ and $\mu_{\mathbb{C}}(A, x) := A \vdash_{\mathrm{cf}} x$. We verify using Lemma 1 that it indeed satisfies the conditions expected of a world-based model.

As before, we prove that interpretation in the model coincides with cut-free provability. The statement needs to be slightly weakened to become provable:
Lemma 2 (Correctness)**.** *The following statements hold:*

- *If $[\![s]\!]_{\mathbb{C}}(A)$ then $A \vdash_{\mathrm{cf}} s$.*
- *If $A \vdash_{\mathrm{ae}} s$ then $[\![s]\!]_{\mathbb{C}}(A)$.*

Proof. We prove the two statements at once by induction on the formula s with A generalised. We have three cases to consider.

- Case x (variable). We have to prove $A \vdash_{\mathrm{cf}} x \Longrightarrow A \vdash_{\mathrm{cf}} x$ – which is immediate – and $A \vdash_{\mathrm{ae}} x \Longrightarrow A \vdash_{\mathrm{cf}} x$ which follows from the second rule.
- Case $\bot$. We have to prove $A \vdash_{\mathrm{cf}} \bot \Longrightarrow A \vdash_{\mathrm{cf}} \bot$ – which is again immediate – and $A \vdash_{\mathrm{ae}} \bot \Longrightarrow A \vdash_{\mathrm{cf}} \bot$ which follows from the second rule.
- Case $s \to t$. This is the interesting case. Let us first recall what $[\![s \to t]\!]_{\mathbb{C}}(A)$ evaluates to:

$$\forall A'.\ A \subseteq A' \wedge [\![s]\!]_{\mathbb{C}}(A') \Longrightarrow [\![t]\!]_{\mathbb{C}}(A') \tag{1}$$

  Let us now look at the two propositions we have to prove.

  – Assuming 1, we have to prove $A \vdash_{\mathrm{cf}} s \to t$. By applying the first rule, it remains to show $s :: A \vdash_{\mathrm{cf}} t$. By induction hypothesis on t, it suffices to show $[\![t]\!]_{\mathbb{C}}(s :: A)$. We now apply 1 which leaves us to prove both that $A \subseteq s :: A$ – which is easy – and $[\![s]\!]_{\mathbb{C}}(s :: A)$. From the induction hypothesis on s this time, it is sufficient to prove $s :: A \vdash_{\mathrm{ae}} s$. We conclude using the fourth rule.

  – Assuming $A \vdash_{\mathrm{ae}} s \to t$ we have to show 1 holds. In other words, we assume $A \subseteq A'$ and $[\![s]\!]_{\mathbb{C}}(A')$ and show $[\![t]\!]_{\mathbb{C}}(A')$. By induction hypothesis on t it is enough to prove $A' \vdash_{\mathrm{ae}} t$. We can conclude from the third rule if we first prove $A' \vdash_{\mathrm{ae}} s \to t$ and $A' \vdash_{\mathrm{cf}} s$. The former we obtain by assumption up to Lemma 1. The latter follows by induction hypothesis. □

We write $[\![A]\!]_M(w)$ for `ctx_winterp` $M\ w\ A$.
Lemma 3. *$[\![A]\!]_{\mathbb{C}}(A)$ holds.*

Proof. By induction on the first occurrence of A. The second occurrence needs to be suitably generalised first. □

Theorem 4 (Cut elimination)**.** *If for all M, w, such that $[\![A]\!]_M(w)$ we have $[\![s]\!]_M(w)$, then we have $A \vdash_{\mathrm{cf}} s$.*

Proof. By soundness, correctness, and Lemma 3. □

We now work towards a proof that double negation elimination does not have a cut-free proof, and thus no proof in minimal natural deduction. We first prove two auxiliary results.
Lemma 5. *There is no s with $[\,] \vdash_{\mathrm{ae}} s$.*

Proof. Assume $[\,] \vdash_{\mathrm{ae}} s$. We need to prove falsity. The proof is by induction on the proof. The first case follows from the induction hypothesis. The second case is an immediate contradiction. □

We define a recursive function $A \longrightarrow s$ as $[\,] \longrightarrow s := s$ and $(t :: A') \longrightarrow s := (t \to (A' \longrightarrow s))$.
Lemma 6. *Let s be the formula `var 0`. For all A we have that $[\neg\neg s] \vdash_{\mathrm{ae}} A \longrightarrow s$ does not hold.*

Proof. Assume $[\neg\neg s] \vdash_{\mathrm{ae}} A \longrightarrow s$. We prove falsity by induction on the proof with A generalised. The first case follows from the induction hypothesis. The second case is a contradiction by case analysis on A. □

We write `ndm A s` as $A \vdash_m s$.

Theorem 7. *Not for all* s, $[\,] \vdash_m \neg\neg s \to s$.

Proof. Assume $[\,] \vdash_m \neg\neg s \to s$ for $s :=$`var` 0. Using soundness and cut-elimination, we have $[\,] \vdash_{\mathrm{cf}} \neg\neg s \to s$. Case analysis on the proof. The second case is trivial because of the lemma proving that there are no *ae* proofs in the empty context. In the first case, we may assume $[\neg\neg s] \vdash_{\mathrm{cf}} s$. By case analysis we only have one possible case, which makes the last lemma with $A := [\,]$ applicable. □

4 Proof terms

We will now introduce proof terms for natural deduction and show normalisation for them, which is yet another way of proving consistency. Mechanising proof terms and normalisation proofs can be hard, mainly because they need to deal with substitution of variables. We circumvent this by introducing *Hilbert systems* (independently introduced by Frege and Hilbert). They are a way to present deduction without having to manage the context. Consequently, their proof terms will not require substitution, leading to a rather elegant strong normalisation proof.

4.1 Hilbert systems

We give the rules for the Hilbert systems below. While there is a context to be able to do global axiomatic assumptions, the context never changes during a proof.

$$\frac{s \in A}{A \vdash_H s} \qquad \frac{A \vdash_H s \to t \qquad A \vdash_H s}{A \vdash_H t} \qquad \frac{}{A \vdash_H s \to t \to s}$$

$$\frac{}{A \vdash_H (s \to t \to u) \to (s \to t) \to s \to u}$$

a. Define an inductive predicate `hil : list form -> form -> Prop` capturing the rules from above. Make sure you turn the first argument into a *parameter*. Declare the notation `A ⊢H s` for `hil A s`.

b. Prove that Hilbert provability implies minimal provability, *i.e.*

```
Lemma hil_ndm A s :
  A ⊢H s -> A ⊢m s.
```

c. Show the following 3 facts:

1. If $A \vdash_H s$ then $A \vdash_H t \to s$.
2. If $A \vdash_H s \to t \to u$ and $A \vdash_H s \to t$ then $A \vdash_H s \to u$.
3. $A \vdash_H s \to s$.

d. Prove that if $s :: A \vdash_H t$ then $A \vdash_H s \to t$. Explain briefly why it is crucial that A is a parameter of the `hil` predicate.

e. Prove that minimal provability implies Hilbert provability, *i.e.*

```
Fact ndm_hil A s :
  A ⊢m s -> A ⊢H s.
```

4.2 Abstract reduction systems

We work abstractly with a reduction relation $R : A \to A \to \mathsf{Prop}$. In Rocq, use a section as follows:

```
From Stdlib Require Import Lia ZArith List.
Require Import ex1.

Section ARS.

  Context {A : Type} (R : A -> A -> Prop).
```

Such a relation is strongly normalising on an element $x : A$ if all reduction paths from x are finite. We can define this notion inductively as follows:

$$\frac{\forall y.\ R\ x\ y \to \mathsf{SNon}\ R\ y}{\mathsf{SNon}\ R\ x}$$

a. Define an inductive relation `SN_on : A -> Prop` with the above rules. Work in the section above.

b. Define the reflexive transitive closure of R, mathematically defined as follows

$$\frac{}{R^*\ x\ x} \qquad \frac{R\ x\ y}{R^*\ x\ y} \qquad \frac{R^*\ x\ y \qquad R^*\ y\ z}{R^*\ x\ z}$$

as an inductive relation `rtc : A -> A -> Prop`.

c. Prove

```
Lemma SN_on_rtc x y :
  SN_on x -> rtc x y -> SN_on y.
```

d. Given a typing relation $T : A \to \mathsf{Prop}$ and a value relation $V : A \to \mathsf{Prop}$, strong normalisation implies the existence of normal forms if R preserves typing and satisfies progress, *i.e.* any well-typed term, either steps or is a value. Formally:

```
Variables T V : A -> Prop.

Variable Hpres : forall x y, T x -> R x y -> T y.
Variable Hprog : forall x, T x -> (exists y, R x y) \/ V x.

Lemma SN_to_WN x :
  T x -> SN_on x -> exists v, rtc x v /\ T v /\ V v.
```

e. You can now close the `ARS` section with `End ARS`. Prove double induction on strong normalisation proofs, *i.e.*

```
Lemma SN_on_double_ind [A B : Type] [R1 : A -> A -> Prop] [R2 : B -> B -> Prop]
          (P : A -> B -> Prop) :
  (forall (a : A) (b : B),
    (forall (a' : A), R1 a a' -> SN_on R1 a') ->
    (forall (a' : A), R1 a a' -> P a' b) ->
    (forall (b' : B), R2 b b' -> SN_on R2 b') ->
    (forall (b' : B), R2 b b' -> P a b') ->
    P a b) ->
  forall (x : A) (y : B), SN_on R1 x -> SN_on R2 y -> P x y.
```

4.3 Combinatory Logic

Combinatory logic was introduced by Schoenfinkel and Curry. It can be seen as exactly the proof terms of a Hilbert system.

```
Inductive term :=
| S | K | V (n : nat) | app (e1 e2 : term).
Coercion app : term >-> Funclass.

(* Import your solution to exercise 1. *)
From Project Require Import ex1.

Section typing.

  Variable A : list form.

  Reserved Notation "⊢ e : s" (at level 60, e at next level).
```

We write $e_1\ e_2$ for `app` $e_1\ e_2$, and you can do it in your code too thanks to the `Coercion` line.

a. Define a typing relation `typing : term -> form -> Prop` as follows:

$$\frac{\mathsf{nth\_error}\ A\ n = \mathsf{Some}\ s}{A \vdash \mathsf{V}\ n : s} \qquad \frac{A \vdash e_1 : s \to t \qquad A \vdash e_2 : s}{A \vdash e_1\ e_2 : t}$$

$$\frac{}{A \vdash \mathsf{K} : s \to t \to s} \qquad \frac{}{A \vdash \mathsf{S} : (s \to t \to u) \to (s \to t) \to s \to u}$$

with notation `Notation "⊢ e : s" := (typing e s) (at level 60, e at next level).`

b. Show that Hilbert system provability and the existence of well-typed proof terms are equivalent:

```
Lemma hil_equiv s :
  A ⊢H s <-> exists e, ⊢ e : s.
```

c. Define a reduction relation `red : term -> term -> Prop` as follows:

$$\frac{}{\mathsf{K}\ e_1\ e_2 \succ e_1} \qquad \frac{}{\mathsf{S}\ e_1\ e_2\ e_3 \succ e_1\ e_3\ (e_2\ e_3)} \qquad \frac{e_1 \succ e_1'}{e_1\ e_2 \succ e_1'\ e_2} \qquad \frac{e_2 \succ e_2'}{e_1\ e_2 \succ e_1\ e_2'}$$

with notation `Notation "e1 ≻ e2" := (red e1 e2) (at level 60).`

d. Show that one-step reduction preserves types, *i.e.*

```
Lemma preservation e1 e2 s :
  ⊢ e1 : s ->
  e1 ≻ e2 ->
  ⊢ e2 : s.
```

e. We define the reflexive transitive closure of reduction as follows.

```
Definition reds :=
  rtc red.

Notation "e1 ≻* e2" := (reds e1 e2) (at level 60).
```

Prove the following congruence lemma for reduction and application:

```
Lemma app_red e1 e1' e2 :
  e1 ≻* e1' ->
  e1 e2 ≻* e1' e2.
```

f. Prove that type preservation also extends to the reflexive transitive closure of reduction, *i.e.*

```
Lemma subject_reduction e1 e2 s :
  ⊢ e1 : s ->
  e1 ≻* e2 ->
  ⊢ e2 : s.
```

g. We now define strongly normalising terms and prove that if an application is strongly normalising, then so is the left-hand term. We close the section first.

```
End typing.

Notation "A ⊢ e : s" := (typing A e s) (at level 60, e at next level).

Notation "t1 ≻ t2" := (red t1 t2) (at level 60).
Notation "t1 ≻* t2" := (reds t1 t2) (at level 60).

Definition SN (e : term) :=
  SN_on red e.

Lemma SN_app e1 e2 :
  SN (e1 e2) -> SN e1.
```

h. We now define so-called neutral terms and show they are closed under application:

```
Definition neutral (e : term) :=
  match e with
  | app K _ | K | app (app S _) _ | S | app S _ => False
  | _ => True
  end.

Lemma neutral_app e1 e2 :
  neutral e1 -> neutral (e1 e2).
```

i. Finally, we prove that well-typed terms either step or are some form of value. In this case, we can define a term to be a value if it is not neutral:

```
Lemma progress e s :
  ([] ⊢ e : s) -> (exists e', red e e') \/ ~neutral e.
```

4.4 Normalisation

For this exercise, we again give a detailed proof on paper. Your task is to formalise all notions in Rocq, and prove the lemmas and theorems. You will not have to invent mathematical arguments or intermediate lemmas: the structure of the proof is given on paper without gaps.

Definition 1. *We define a notion of semantic typing* $\vDash e : s$ *as a recursive function on* s*:*

$$\vDash e : \bot := \mathsf{SN}\ e \qquad \vDash e : \mathsf{var}\ x := \mathsf{SN}\ e \qquad \vDash e : s_1 \to s_2 := \forall e_1.\ \vDash e_1 : s_1 \Longrightarrow \vDash e\ e_1 : s_2$$

The relation for semantic typing is often called a *logical* relation.

Theorem 8. *The following holds for all* s *and* e*:*

1. *If* $\vDash e : s$ *then* $\mathsf{SN}\ e$*.*
2. *If* $\vDash e : s$ *then for all* e' *with* $e \succ^* e'$ *it holds that* $\vDash e' : s$*.*
3. *If* e *is neutral and for all* e' *with* $e \succ e'$ *we have* $\vDash e' : s$*, it holds that* $\vDash e : s$*.*

Proof. By induction on s with e generalised.

We have three cases, but the proof is the same for $s = \mathsf{var}\ x$ and $s = \bot$. (1) is trivial. (2) follows from `SN_on_rtc`. (3) is by definition of strong normalisation.

Now let $s = s_1\ s_2$. We have three induction hypotheses each for s_1 and s_2 and three things to prove.

For (1), we can assume that e semantically has type $s_1 \to s_2$, *i.e.* that for all e_1 with $\vDash e_1 : s_1$ we have $\vDash e\ e_1 : s_2$. We need to prove that e strongly normalises. We use `SN_app` with $e_2 := \mathsf{V}\ 0$ (or any other variable), so we have to prove that $\mathsf{SN}\ (e\ (\mathsf{V}\ 0))$. We use the induction hypothesis (1) for s_2 and have to prove $\vDash e\ (\mathsf{V}\ 0) : s_2$. By assumption, it suffices to prove $\vDash \mathsf{V}\ 0 : s_1$. We use the induction hypothesis (3) for s_1. $\mathsf{V}\ 0$ is neutral and does not reduce to anything, so we are done.

For (2), assume that e semantically has type $s_1 \to s_2$, *i.e.* that for all e_1 with $\vDash e_1 : s_1$ we have $\vDash e\ e_1 : s_2$ and that $e \succ^* e'$. We need to prove that e' semantically has type $s_1 \to s_2$, *i.e.* assume e_1 with $\vDash e_1 : s_1$ and need to prove that $\vDash e'\ e_1 : s_2$. We use the induction hypothesis (2) for s_2 and that $\vDash e_1 : s_1$. It suffices to prove that $e\ e_1 \succ^* e'\ e_1$, which follows from `app_red`.

For (3), assume that e is neutral and for all e' with $e \succ e'$ we have $\vDash e' : s_1 \to s_2$ (call this assumption H). We need to prove that e semantically has type $s_1 \to s_2$, *i.e.* assume e_1 with $\vDash e_1 : s_1$ and need to prove that $\vDash e\ e_1 : s_2$. We use the induction hypothesis (1) for s_1 on the assumption to derive $\mathsf{SN}\ e_1$. Now the proof is by induction on this strong normalisation proof, *i.e.* we can assume that for any e_1' with $e_1 \succ e_1'$ with $\vDash e_1' : s_1$ we have that $\vDash e\ e_1' : s_2$. We use the induction hypothesis (3) for s_2. We have to prove that $e\ e_1$ is neutral, which follows from e being neutral from `neutral_app`, and then assume e' with $ee_1 \succ e'$ and have to prove that $\vDash e' : s_2$. We do a case analysis on $e\ e_1 \succ e'$. There are four cases:

- For $e = \mathsf{K}\ e_3$, we have a contradiction because $\mathsf{K}\ e_3$ is not neutral.
- For $e = \mathsf{S}\ e_3\ e_4$, we have a contradiction because $\mathsf{S}\ e_4\ e_5$ is not neutral.
- For $e \succ e_3$, we can use H and are done.
- For $e_1 \succ e_1'$, we first use the induction hypothesis for e_1, and it suffices to prove that $\vDash e_1' : s_1$. This follows from using the induction hypothesis (2) for s_1, because $\vDash e_1 : s_1$ and $e_1 \succ^* e_1'$ follows from $e_1 \succ e_1'$.

□

Lemma 9. *For any s and t, $\vDash \mathsf{K} : s \to t \to s$*

Proof. By definition, we can assume $\vDash e_1 : s$ and $\vDash e_2 : t$ and have to prove $\vDash \mathsf{K}\ e_1\ e_2 : s$. By application of Theorem 8, we know that e_1 and e_2 are strongly normalising. We use double induction on strong normalisation, proved above.

We use Theorem 8, point (3), to prove $\vDash \mathsf{K}\ e_1\ e_2 : s$. We need to show that $\mathsf{K}\ e_1\ e_2$ is neutral – which follows by definition, It remains to show that any e' with $\mathsf{K}\ e_1\ e_2 \succ e'$ fulfills $\vDash e' : s$. There are three cases:

1. $\mathsf{K}\ e_1\ e_2 \succ e_1$. $\vDash e_1 : s$ follows by assumption.
2. $\mathsf{K}\ e_1\ e_2 \succ e'\ e_2$ via $\mathsf{K}\ e_1 \succ e'$. This cannot arise from K stepping, so it has to be that $e' = \mathsf{K}\ e_1'$ and $e_1 \succ e_1'$. We have to prove $\vDash \mathsf{K} e_1'\ e_2$.

   We can apply the induction hypothesis for e_1 with $e_1 \succ e_1'$. We need to prove $\vDash e_1' : s$, which follows from Theorem 8 point (2).
3. $\mathsf{K}\ e_1\ e_2 \succ \mathsf{K}\ e_1\ e_2'$ via $e_2 \succ e_2'$.

   We can apply the induction hypothesis for e_2 with $e_2 \succ e_2'$. We need to prove $\vDash e_2' : t$, which follows from Theorem 8 point (2).

□

Lemma 10. *For any s, t, u, $\vDash \mathsf{S} : (s \to t \to u) \to (s \to t) \to s \to u$.*

Proof. Essentially the same proof as for K. But one first needs to state and prove a *triple* induction principle for strong normalisation. □

We recommend proving the lemma about S *last*, *i.e.* stating and admitting it first.

Theorem 11. *Assume that whenever the n-th element of A is s, $\vDash \mathsf{V}\ n : s$ holds. Then $A \vdash e : s$ implies $\vDash e : s$.*

Proof. By induction on typing. The variable case uses the assumption. The K and S cases follow from the last two lemmas. The application case follows from the induction hypotheses. □

Lemma 12. *Any term well typed in the empty context is strongly normalising.*

Proof. Straightforward from the last two theorems. □

Lemma 13. *Any well-typed term e in the empty context reduces to a term e' of the same type which is not neutral.*

Proof. Follows from `SN_to_WN`. □

4.5 Consistency

We are going to prove that the Hilbert system is consistent by proving that there is no normal term of type $\bot$.

a. Prove that there is no term of type `bot` in the empty context.

```
Lemma noterm e :
  ~ [] ⊢ e : bot.
```

Use weak normalisation first and then do case analysis on the proof that the value is not neutral until the goal is proved.

b. Prove that intuitionistic natural deduction is consistent, *i.e.*

```
Corollary nd_consistent :
  ~ [] ⊢m bot.
```

c. Prove that classical natural deduction is consistent, *i.e.*

```
Corollary ndc_consistent :
  ~ [] ⊢c bot.
```

5 Conjunction and disjunction

In this exercise, you will redo exercises 1-4 for a more expressive logic. Please make sure to submit the original files for exercises 1-4 and separate files with the extensions asked in this exercise.

a. Extend the formula type by conjunction and reprove all results.

b. Extend the formula type by disjunction and reprove all results. You will get stuck on defining semantic typing for disjunction. Ask us for hints!

c. Can you use cut-elimination / normalisation to deduce results other than consistency and unprovability of double negation elimination?