

Initiation aux preuves formelles CM1

Pierre Rousselin

Université Sorbonne Paris Nord

Septembre 2025

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Cours et ressources

- ▶ Trois heures par semaine le mardi matin
- ▶ Chaque semaine :
 - ▶ TP (5) ou TD (4) ou CM (2) la première heure et demi
 - ▶ TP la seconde heure et demi
- ▶ Rien la semaine du 20 octobre (vacances) et la semaine du 27 octobre (partiels).
- ▶ Page sur moodle pour les rendus
- ▶ Lien vers la page principale du cours :
`https://www.math.univ-paris13.fr/~rousselin/ipf2.html`
- ▶ Responsable du cours et TP groupe DLA : Pierre Rousselin
`rousselin@univ-paris13.fr`
- ▶ Chargée de TP groupe DLB : Micaela Mayero (LIPN, équipe logique)
`mayero@lipn.univ-paris13.fr`

Note du module

- ▶ Évaluation continue (EvC) : plusieurs devoirs à rendre, surtout (seulement ?) des fichiers à rendre
- ▶ TP noté d'une heure et demi (CTP), probablement le 9 décembre
- ▶ Épreuve écrite (Pa) la semaine du 15 décembre (2h)
- ▶ Note à la fin du semestre :

$$N_{\text{ipf}} = \frac{\text{EvC} + \text{CTP} + \text{Pa}}{3}$$

Note du module

- ▶ Évaluation continue (EvC) : plusieurs devoirs à rendre, surtout (seulement ?) des fichiers à rendre
- ▶ TP noté d'une heure et demi (CTP), probablement le 9 décembre
- ▶ Épreuve écrite (Pa) la semaine du 15 décembre (2h)
- ▶ Note à la fin du semestre :

$$N_{\text{ipf}} = \frac{\text{EvC} + \text{CTP} + \text{Pa}}{3}$$

- ▶ Si $N_{\text{ipf}} < 10$ et passage au niveau 2, alors épreuve de seconde chance *sur demande* sous la forme d'un TP noté (CTP') la semaine du 6 juin

$$N'_{\text{ipf}} = \sup \{ N_{\text{ipf}}, \text{CTP}' \}$$

- ▶ UE complémentaire (non fondamentale) qui vaut 2 ECTS

But du module

- ▶ Travailler sur la notion de preuve mathématique
- ▶ sous sa forme la plus rigoureuse, sur machine (preuve formelle)
- ▶ Intérêt en soi (applications réelles dans l'industrie des logiciels sûrs)
- ▶ Intérêt pour se muscler en mathématiques : comment rédiger une démonstration très détaillée, acquérir des réflexes de raisonnement et quelques bases d'algèbre et d'analyse
- ▶ Module dans sa 5^{ème} année, assez unique à ce niveau et toujours un peu expérimental.
- ▶ Vu à la $\mathcal{TV}$ gazette des mathématiciens :
<https://hal.science/hal-03979238>

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Assistant de preuve

- ▶ Programme informatique permettant la vérification de preuves mathématiques
 - ▶ pour les mathématiciens ($\nearrow$ mais encore faible en proportion)
 - ▶ pour les propriétés de programmes informatiques (depuis de nombreuses années déjà, mais en constante progression)
- ▶ Le premier : Automath de Nicolas de Bruijn, TU Eindhoven, 1966
- ▶ Mizar, Andrzej Trybulec, Université de Varsovie, 1973
- ▶ Coq/Rocq, Thierry Coquand, Gerard Huet, Christine Paulin-Mohring, Inria, 1984
- ▶ HOL-Isabelle, Lawrence Paulson, Cambridge, Munich, 1986
- ▶ Lean, Leonardo de Moura, Microsoft Research, 2013

Le seul qui n'est plus développé est Automath, chacun a ses avantages et ses inconvénients.

Coq/Rocq

- ▶ Nommé d'après Thierry Coquand, inventeur du « Calculus of Constructions » (COC, d'où aussi Coq), une variante de la théorie des types de Per Martin-Löf qui est la base théorique de Coq.
- ▶ Renommé Rocq (comme Rocquencourt, siège historique de l'Inria) en 2025 car en anglais un homophone de « Coq » peut être considéré comme grossier.
- ▶ Quelques traits de Coq/Rocq :
 - ▶ prend le calcul très au sérieux, beaucoup de théorèmes peuvent être simplement obtenu « par calcul » ;
 - ▶ prend sa cohérence interne très au sérieux, on n'ajoute des fonctionnalités seulement lorsqu'il est prouvé (formellement ou non) qu'elles préservent les propriétés clé du système ;
 - ▶ prend la « compatibilité ascendante » très au sérieux (les développements faits avec les anciennes versions de Coq devraient fonctionner avec les versions plus récentes).
 - ▶ revers de la médaille : on doit donc composer avec certains mauvais choix « historiques » qui encombrant l'assistant.

Quelques projets spectaculaires en Coq

- ▶ CompCert, un compilateur C prouvé correct en Coq :
<https://compcert.org/>
(Xavier Leroy et collaborateurs, 2005 – aujourd’hui)
- ▶ Fiat Cryptography, un générateur de code pour la cryptographie prouvé en Coq : <https://github.com/mit-plv/fiat-crypto>
(2017 Adam Chlipala, Andres Erbsen, Jason Gross, ..., 2017 – aujourd’hui)
- ▶ La preuve formelle du théorème des 4 couleurs (44000 loc) :
<https://github.com/coq-community/fourcolor>
(2005 Georges Gonthier et Benjamin Werner)
- ▶ La preuve formelle du théorème de Feit-Thompson (40000 loc) :
<https://github.com/math-comp/odd-order>
(2012 Georges Gonthier et de nombreux collaborateurs)

Coq en pratique pour vous

- ▶ Vous écrirez essentiellement des preuves en utilisant des commandes appelées *tactiques* qui modifient l'état de la preuve jusqu'à ce qu'il n'y ait plus rien à prouver.
- ▶ Nos tactiques sont des tactiques de base de Coq.
- ▶ Un fichier de script Coq est un *fichier texte*, encodé en UTF-8, dont le nom se termine par `.v` (comme *Vernaculaire*, nom donné au langage de commandes de Coq).
- ▶ Vous pouvez installer Coq :
 - ▶ avec la *Coq Platform* :
`https://github.com/coq/platform/releases`
 - ▶ sous Linux ou Mac avec `opam` :
`https://coq.inria.fr/opam-using.html`
 - ▶ sous Linux avec votre gestionnaire de paquets, par exemple (Debian, Ubuntu, Mint, ...) `sudo apt install coq`
- ▶ Mais vous pouvez aussi utiliser simplement notre interface web `jsCoq` sur le site du cours.

Coq en pratique pour vous (2)

Il faut aussi une interface pratique pour modifier les fichiers texte tout en voyant l'état de la preuve évoluer.

- ▶ Interface web `jsCoq` : on espère améliorer cette interface au cours du semestre. Pour l'instant, la sauvegarde est expérimentale.
- ▶ CoqIDE : l'IDE (*Integrated Development Environment*) de base encore développé par l'Inria, fait le job, au prix d'un éditeur de texte assez pauvre (avec des bugs sur les Mac récents)
- ▶ Emacs avec l'extension *Proof General*, sans doute la solution la plus mature
- ▶ Visual Studio Code (de Microsoft, mais libre, mais de Microsoft) avec l'extension `vsCoq` ou `vsCoq2` (dépend de votre version de Coq) ; l'extension a plus de fonctionnalités que les autres
- ▶ Vim ou Neovim avec l'extension *CoqTail*, l'extension n'est pas très riche, mais au moins on est dans Vim

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Le type `Prop` des propositions

- ▶ Le type `Prop` contient les *propositions* (qui sont eux-mêmes des types) qu'on va chercher à prouver ou réfuter.
- ▶ Par exemple, $2 + 2 = 4$, $2 + 2 = 5$ et $3 < 2$ sont des propositions, qu'elles soient prouvables ou non.
- ▶ Si A et B sont des propositions, on peut former les propositions :
 - ▶ $A \rightarrow B$ (A implique B)
 - ▶ $A \wedge B$ (A et B)
 - ▶ $A \vee B$ (A ou B)
 - ▶ $\sim A$ (non A)
 - ▶ $A \leftrightarrow B$ (A équivaut à B)qu'on expliquera par la suite.

Le type Prop des propositions

- Ci-dessous, la commande **Check** imprime le type d'un terme (ici, toujours **Prop**).
- Les variables A et B sont locales à la **Section**.

```
Check 2 + 2 = 4. (* Prop *)
```

```
Check 2 + 2 = 5. (* Prop *)
```

```
Check 3 < 2. (* Prop *)
```

```
Section ConnecteursLogiques.
```

```
Variables (A B : Prop).
```

```
Check (A -> B). (* Prop *)
```

```
Check (A /\ B). (* Prop *)
```

```
Check (A \/ B). (* Prop *)
```

```
Check (~ A). (* Prop *)
```

```
Check (A <-> B). (* Prop *)
```

```
End ConnecteursLogiques.
```

Le type Prop des propositions

- ▶ On ne donnera aucune définition :
 - ▶ du type **Prop** lui-même et
 - ▶ des connecteurs logiques ci-dessus en Coq,
- ▶ car cela serait bien trop abstrait et compliqué à ce niveau (et franchement ce n'est pas notre but dans ce cours, même si c'est passionnant).
- ▶ On adopte le point de vue suivant : à chaque connecteur est associé de une à trois règles logiques qui chacune ont un effet *mécanique* sur l'état de la preuve.
- ▶ Ces règles ont toujours la forme suivante :
 - ▶ **Comment l'utiliser ?** (Le mot savant est « éliminer. »)
 - ▶ **Comment le prouver ?** (Le mot savant est « introduire. »)

Premier exemple

```
Lemma imp_refl : forall (P : Prop), P -> P.
```

```
Proof.
```

```
  intros P. (* Soit P une proposition. *)
```

```
  intros H. (* On suppose que P est vraie (hypothèse (H)). *)
```

```
  exact H. (* Il faut prouver que P est vraie, or c'est le cas d'après
```

```
Qed.
```

On voit ici 3 choses :

- ▶ La règle d'introduction du « pour tout » $\forall$: pour prouver un « pour tout » on commence par « Soit » et on introduit une nouvelle variable, sans hypothèse sur celle-ci. Ici, c'est fait avec `intros P`.
- ▶ La règle d'introduction de l'implication : pour prouver une implication $A \implies B$, on suppose que A est vrai et on cherche à prouver B sous cette hypothèse. C'est encore `intros` qui est utilisée.
- ▶ Comment conclure en Coq lorsque le contexte contient une preuve du but à prouver. Il suffit d'utiliser la tactique `exact` avec comme argument le nom de la preuve du contexte à utiliser.

Premier exemple avec but

```
Lemma imp_refl :  
  forall (P : Prop), P -> P.  
Proof.
```

```
  intros P.
```

```
  intros H.
```

```
  exact H.  
Qed.
```

```
===== (1 / 1)
```

```
forall P : Prop, P -> P
```

```
P : Prop
```

```
===== (1 / 1)
```

```
P -> P
```

```
P : Prop
```

```
H : P
```

```
===== (1 / 1)
```

```
P
```

```
All goals completed.
```

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Introduction du « pour tout »

En logique, on écrit souvent les règles sous la forme de *règles d'inférence*. Ici, la règle d'introduction du pour tout, qu'on note $\forall I$.

$$\frac{\Gamma, x : T \vdash P}{\Gamma \vdash \forall x : T, P} (\forall I)$$

- ▶ Le symbole $\vdash$ (qu'on lit en français « taquet ») signifie que ce qui est à droite peut se déduire de ce qui est à gauche.
- ▶ Ici, Γ désigne un *contexte* quelconque (donc un ensemble d'hypothèses).
- ▶ Au dessus de la barre verticale, on trouve les prémisses, et en dessous la conclusion.
- ▶ Donc, de haut en bas : si dans le contexte Γ avec en plus l'hypothèse que x est de type T (et aucune autre hypothèse sur x), je sais prouver P ; alors dans le contexte Γ , je sais prouver $\forall x : T, P$.
- ▶ Et de bas en haut, pour prouver à partir de Γ la formule $\forall x : T, P$, il suffit de prouver P à partir de Γ auquel on ajoute l'hypothèse $x : T$ (et aucune autre hypothèse sur x).

Introduction du « pour tout »

devient

contexte

===== (1 / 1)

forall $x : T$, P

contexte

$x : T$

===== (1 / 1)

P

- ▶ En logique : en appliquant la règle d'introduction du « pour tout » $\forall I$.
- ▶ En Coq, avec la tactique **intros** x .
- ▶ En mathématiques, avec la phrase « Soit x de type T », par exemple « Soit x un nombre réel ».

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Introduction du « implique »

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \implies B} (\implies \text{I})$$

- ▶ De haut en bas : si dans le contexte Γ avec en plus l'hypothèse que A est vrai (et aucune autre hypothèse supplémentaire), je sais prouver B ; alors dans le contexte Γ , je sais prouver $A \implies B$.
- ▶ De bas en haut, pour prouver à partir de Γ la formule $A \implies B$, il suffit de prouver B à partir de Γ auquel on ajoute l'hypothèse que A est vraie (et aucune autre hypothèse).

Introduction du « implique »

devient

contexte

===== (1 / 1)

A -> B

contexte

H : A

===== (1 / 1)

B

- ▶ En logique : en appliquant la règle d'introduction du « implique »
 $\Rightarrow$ I.
- ▶ En Coq, avec la tactique **intros** H. (Remarquer qu'on nomme l'hypothèse !)
- ▶ En mathématiques, avec la phrase « On suppose que A est vraie. »

Modus Ponens

La règle d'élimination (utilisation) de « implique » est le *modus ponens*. En Coq, il correspond à la tactique **apply**.

Lemma `modus_ponens` : **forall** (P Q : **Prop**), P -> (P -> Q) -> Q.

Proof.

```
  intros P Q. (* Soient P et Q deux propositions. *)
```

```
  intros H H'. (* On suppose (hypothèse (H)) que P est vraie et  
                (hypothèse (H')) que (P -> Q). *)
```

```
  apply H'. (* Par (H'), pour prouver Q, il suffit de prouver P *)
```

```
  exact H. (* Or, l'hypothèse H fournit une preuve de P. *)
```

Qed.

Modus Ponens avec but

Lemma modus_ponens :

forall (P Q : Prop), P -> (P -> Q) -> Q.

Proof.

===== (1 / 1)

forall (P Q : Prop), P -> (P -> Q) -> Q.

intros P Q.

P, Q : Prop

===== (1 / 1)

P -> (P -> Q) -> Q

intros H H'.

P, Q : Prop

H : P

H' : P -> Q

===== (1 / 1)

Q

apply H'.

P, Q : Prop

H : P

H' : P -> Q

===== (1 / 1)

P

exact H.

Qed.

All goals completed.

Élimination du « implique »

$$\frac{\Gamma \vdash A \quad \Gamma \vdash A \implies B}{\Gamma \vdash B} (\implies E)$$

- ▶ De haut en bas : si dans le contexte Γ je sais prouver A et $A \implies B$, alors dans le contexte Γ , je sais prouver B .
- ▶ De bas en haut, pour prouver B partir de Γ il suffit de prouver A et $A \implies B$ à partir de Γ .

Élimination du « implique »

devient

reste du contexte

$H : A \rightarrow B$

===== (1 / 1)

B

reste du contexte

$H : A \rightarrow B$

===== (1 / 1)

A

- ▶ En logique : en appliquant la règle d'élimination du « implique » $\Rightarrow E$.
- ▶ En Coq, avec la tactique **apply** H .
- ▶ En mathématiques, avec la phrase « Comme on sait que A implique B , pour prouver B , il suffit de prouver A . »
- ▶ Attention, utiliser le modus ponens avec une certaine implication est **un choix** qui peut être pertinent ou non !

Attention au **apply** non réfléchi

Lemma plop : forall (P Q R : **Prop**),

P -> (P -> R) -> (Q -> R) -> R.

Proof.

intros P Q R H H1 H2.

apply H2.

donne l'état de preuve suivant :

P, Q, R : **Prop**

H : P

H1 : P -> R

H2 : Q -> R

===== (1 / 1)

Q

Qu'en pensez-vous ?

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Règle d'introduction du « et »

Lemma conj : forall (P Q : Prop), P -> Q -> P /\ Q.

Proof.

intros P Q H H'.

(Soient P et Q deux proposition telles que :*

(H) P est vraie

*(H') Q est vraie. *)*

split. *(* Pour montrer P /\ Q... *)*

- *(* ... on montre d'abord que P est vraie,... *)*

exact H. *(* ...ce qui est le cas d'après (H), ...*)*

- *(* ... puis que Q est vraie, ... *)*

exact H'. *(* ... ce qui est le cas d'après (H'). *)*

Qed.

- Pour **prouver** un « et » on utilise **split**.
- Les “bullets” permettent de structurer la preuve. On peut utiliser
 - ou + ou * ou -- ou ++, etc.

Introduction du « et »

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} (\wedge\text{I})$$

- ▶ De haut en bas : si dans le contexte Γ je sais prouver A et B , alors dans le contexte Γ , je sais prouver $A \wedge B$.
- ▶ De bas en haut, pour prouver $A \wedge B$ à partir de Γ il suffit de prouver A à partir de Γ et B à partir de Γ .

Introduction du « et »

devient

contexte
===== (1 / 1)
A /\ B

contexte
===== (1 / 2)
A

contexte
===== (2 / 2)
B

- ▶ En logique : en appliquant la règle d'introduction du « et » $\wedge I$.
- ▶ En Coq, avec la tactique **split**.
- ▶ En mathématiques, avec la phrase « On commence par prouver $A...$ » puis après la preuve de A , « On prouve maintenant $B ...$ ». L'essentiel est qu'il soit facile de voir l'organisation de la preuve.

Élimination du « et »

Lemma proj1 : forall (P Q : Prop), P /\ Q -> P.

Proof.

intros P Q H. (*Soient P et Q deux propositions telles
que (H) : P /\ Q est vraie. **)

destruct H **as** [H1 H2]. (*De (H) on déduit que
(H1) : P est vraie et
(H2) : Q est vraie. **)

exact H1. (*On veut prouver P, or (H1) est une preuve de P. **)

Qed.

- ▶ En Coq, tactique **destruct** H **as** [H1 H2] .
- ▶ Remarquer les crochets et les nouveaux noms pour les deux nouvelles hypothèses.
- ▶ Ceci supprime l'hypothèse H du contexte, en effet, on l'a déjà exploitée il est peu probable qu'on en aura encore besoin.

Élimination du « et »

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} (\wedge\text{E-g})$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} (\wedge\text{E-d})$$

- ▶ De haut en bas : si dans le contexte Γ je sais prouver $A \wedge B$, alors dans le contexte Γ , je sais prouver A et B (oui, ça a l'air un peu bête ce qu'on raconte).
- ▶ De bas en haut, pour prouver $A \wedge B$ à partir de Γ il suffit de prouver A à partir de Γ et B à partir de Γ .

Élimination du « et »

devient

reste du contexte

H : A /\ B

===== (1 / 1)

but

reste du contexte

H1 : A

H2 : B

===== (1 / 2)

but

- ▶ En logique : en appliquant les deux règles d'élimination du « et » $\wedge$ I-g et $\wedge$ I-d.
- ▶ En Coq, avec la tactique **destruct** H **as** [H1 H2] .
- ▶ En mathématiques, ... en général on ne détaille pas cette étape, même lorsqu'on est très pointilleux. On dira plutôt directement : « On suppose (hypothèse H1) que A est vraie et (hypothèse H2) que B est vraie. »

Intro pattern

- Pour se rapprocher de l'usage mathématique (et aussi gagner du temps) on peut remplacer `intros H. destruct H as [H1 H2].` par `intros [H1 H2].` directement.
- Cela s'appelle un *intro pattern* en Coq.

`Lemma proj1 : forall (P Q : Prop), P /\ Q -> P.`

`Proof.`

```
  intros P Q [H1 H2]. (* Soient P et Q deux propositions telles
                        (H1) : P est vraie et
                        (H2) : Q est vraie. *)
```

```
  exact H1. (* On veut prouver P, or (H1) est une preuve de P. *)
```

`Qed.`

Associativité à droite

- ▶ Il est clair mathématiquement que pour toutes propositions A , B et C , la proposition $(A \text{ et } (B \text{ et } C))$ est équivalente à $((A \text{ et } B) \text{ et } C)$, et on ne fait jamais la distinction entre les deux.
- ▶ Malheureusement ceci ne peut pas être le cas en Coq, où l'associativité de $\wedge$ est un théorème.
- ▶ Il faut donc faire un choix lorsqu'on écrit $A \wedge B \wedge C$ et ce choix est arbitraire. Les concepteurs de Coq ont choisi, comme pour l'implication de dire que pas de parenthèse équivaut pour $\wedge$ à $A \wedge (B \wedge C)$, on dit que $\wedge$ est *associatif à droite*.

Associativité à droite

```
Lemma and_32 : forall (A B C : Prop), A /\ B /\ C -> B.
```

```
Proof.
```

```
  intros A B C H.
```

```
  destruct H as [H1 H2].
```

```
  destruct H2 as [H2 H3]. (* Je peux réutiliser H2. *)
```

```
  exact H2.
```

```
Qed.
```

```
Lemma and_32' : forall (A B C : Prop), A /\ B /\ C -> B.
```

```
Proof.
```

```
  intros A B C H.
```

```
  destruct H as [H1 [H2 H3]].
```

```
  exact H2.
```

```
Qed.
```

```
Lemma and_32'' : forall (A B C : Prop), A /\ B /\ C -> B.
```

```
Proof.
```

```
  intros A B C [H1 [H2 H3]].
```

```
  exact H2.
```

```
Qed.
```

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Règles d'introduction du « ou »

Lemma or_introl : forall (A B : Prop), A -> (A \\/ B).

Proof.

intros A B H.

left. (* On veut prouver A \\/ B, on va prouver A. *)

exact H.

Qed.

Lemma or_intror : forall (A B : Prop), B -> (A \\/ B).

Proof.

intros A B H.

right. (* On veut prouver A \\/ B, on va prouver B. *)

exact H.

Qed.

- ▶ Pour prouver un « ou » on utilise left ou right.
- ▶ C'est un choix important qui ne doit pas être fait au hasard.
- ▶ Ne pas hésiter à revenir en arrière lorsqu'on voit, en regardant l'état de la preuve, que l'on est coincé.

Introductions du « ou »

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \text{ (}\vee\text{I-g)}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \text{ (}\vee\text{I-d)}$$

- ▶ De haut en bas : si dans le contexte Γ je sais prouver A ou, si dans le contexte Γ , je sais prouver B , alors dans le contexte Γ , je sais prouver $A \vee B$.
- ▶ De bas en haut, pour prouver $A \vee B$ à partir de Γ il suffit de prouver A à partir de Γ ou B à partir de Γ .

Élimination du « ou »

```
Lemma or_ind : forall (A B P : Prop),  
  (A -> P) -> (B -> P) -> A \\/ B -> P.
```

Proof.

```
  intros A B P H1 H2 H.  
  destruct H as [H' | H''].  
  (* On raisonne par disjonction de cas sur A \\/ B. *)  
  - (* Premier cas : On suppose (hypothèse (H')) que A est vraie *)  
    apply H1. (* Dans ce cas on peut appliquer l'implication H1. *)  
    exact H'.  
  - (* Second cas : On suppose (hypothèse (H'')) que B est vraie *)  
    apply H2. (* Dans ce cas on peut appliquer l'implication H2. *)  
    exact H''.
```

Qed.

- ▶ En Coq, tactique **destruct** H **as** [H' | H''] pour un raisonnement par disjonction de cas.
- ▶ Remarquer les crochets et la barre | qui signifie souvent « ou » en informatique. On a ici utilisé des nouveaux noms, mais ce n'était pas nécessaire.
- ▶ Ceci supprime l'hypothèse H du contexte, en effet, on l'a déjà exploitée il est peu probable qu'on en aura encore besoin.

Élimination du « ou »

$$\frac{\Gamma, A \vdash P \quad \Gamma, B \vdash P}{\Gamma, A \vee B \vdash P} (\vee E)$$

- ▶ De haut en bas : si dans le contexte Γ, A je sais prouver P , et dans le contexte Γ, B , je sais aussi prouver P , alors dans le contexte $\Gamma, A \vee B$, je sais prouver P .
- ▶ De bas en haut, pour prouver P à partir du contexte $\Gamma, A \vee B$, il suffit de prouver P à partir de Γ, A et P à partir de Γ, B , autrement dit de se placer dans le cas où A est vrai et dans le cas où B est vrai.

Élimination du « ou »

devient

contexte

$H : A \vee B$

===== (1 / 1)

P

contexte

$H' : A$

===== (1 / 2)

P

contexte

$H'' : B$

===== (2 / 2)

P

- ▶ En logique : en appliquant la règle d'élimination du « ou » $\vee E$.
- ▶ En Coq, avec la tactique **destruct** H **as** [H' | H''] .
- ▶ En mathématiques, avec :
 - ▶ On raisonne par disjonction de cas sur l'hypothèse (H).
 - ▶ Supposons d'abord que A est vraie ... (preuve du but).
 - ▶ Supposons maintenant que B est vraie ... (preuve du but).

La structure de la preuve doit être claire.

Intro pattern

- ▶ Pour gagner du temps, on peut remplacer
`intros H. destruct H as [H' | H''] .` par
`intros [H' | H''] .` directement.
- ▶ C'est une autre forme (disjonctive) d'*intro pattern*. La preuve par cas suit directement. On ne perd rien à faire ça, sauf parfois des redondances dans les deux preuves.

`Lemma or_ind' : forall (A B P : Prop),
 (A -> P) -> (B -> P) -> A \/ B -> P.`

`Proof.`

```
  intros A B P H1 H2 [H' | H''] .  
  - apply H1.  
    exact H' .  
  - apply H2.  
    exact H'' .
```

`Qed.`

Associativité à droite

- ▶ Le $\backslash/$ est encore associatif (preuve plus loin), mais comme c'est un théorème, encore une fois il faut décider ce que signifie $A \backslash/ B \backslash/ C$ en Coq.
- ▶ En Coq, le $\backslash/$ est, comme $/\backslash$ et $\rightarrow$ associatif à droite, donc $A \backslash/ B \backslash/ C$ signifie en fait $A \backslash/ (B \backslash/ C)$ (même si les deux possibilités sont clairement équivalentes).
- ▶ En pratique, cela signifie que pour faire une preuve par cas sur une hypothèse H du type $A \backslash/ B \backslash/ C$, on écrira quelque chose comme **destruct** H **as** $[H1 \mid [H2 \mid H3]]$ ou bien avec un intro pattern, **intros** $[H1 \mid [H2 \mid H3]]$.

Associativité du $\vee$

```
Lemma or_assoc_subproof : forall (A B C : Prop),  
  A  $\vee$  B  $\vee$  C -> (A  $\vee$  B)  $\vee$  C.
```

```
Proof.
```

```
  intros A B C [H1 | [H2 | H3]].  
  - (* On suppose que A est vraie. *)  
    left. left. exact H1.  
  - (* On suppose que B est vraie. *)  
    left. right. exact H2.  
  - (* On suppose que C est vraie. *)  
    right. exact H3.
```

```
Qed.
```


Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Élimination (utilisation) du faux

La proposition `False` en Coq désigne une proposition absurde, qui, lorsqu'elle apparaît dans les hypothèses, met fin à toute démonstration (prouve tout).

```
Lemma ex_falso_quod_libet : forall (P : Prop), False -> P.
```

```
Proof.
```

```
  intros P H.
```

```
  destruct H. (* boum ! prouve tout *)
```

```
Qed.
```

En Coq, `destruct` H, où H est une preuve de `False` termine toujours la preuve (ou sous-preuve) en cours.

Élimination (utilisation) du faux

La proposition `False` en Coq désigne une proposition absurde, qui, lorsqu'elle apparaît dans les hypothèses, met fin à toute démonstration (prouve tout).

```
Lemma ex_falso_quod_libet : forall (P : Prop), False -> P.
```

```
Proof.
```

```
  intros P H.
```

```
  destruct H. (* boum ! prouve tout *)
```

```
Qed.
```

En Coq, `destruct` H, où H est une preuve de `False` termine toujours la preuve (ou sous-preuve) en cours. On peut même réduire la preuve avec un *intro pattern* vide.

```
Lemma ex_falso_quod_libet' : forall (P : Prop), False -> P.
```

```
Proof.
```

```
  intros P []. (* Boum ! *)
```

```
Qed.
```

Ex falso quodlibet

- ▶ En logique, correspond à la règle *ex falso quodlibet*, qui signifie en latin « du faux, on peut déduire ce qu'on veut ».
- ▶ On appelle aussi cette règle le « principe d'explosion ».
- ▶ En logique, le faux est souvent noté $\perp$ (*bottom*).
- ▶ La règle d'élimination est la suivante :

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash P} (\perp E)$$

- ▶ Souvent utilisé pour des cas impossibles dans des preuves par cas.
- ▶ Noter qu'il n'y a pas de règle d'introduction pour $\perp$ (sinon cela voudrait dire qu'on a un moyen de prouver $\perp$ et donc... n'importe quoi).

La tactique `exfalso`

Il arrive qu'un contexte soit incohérent (on dit aussi impossible ou contradictoire). Dans ce cas, on voudrait prouver `False` plutôt que le but actuel, et ce serait un raisonnement valide vu que de `False` on déduit n'importe quoi. La tactique `exfalso` fait précisément ceci : remplacer le but par `False`.

```
Lemma absurd : forall (A P : Prop), A -> (A -> False) -> P.
```

```
Proof.
```

```
  intros A P H H'.
```

```
  exfalso. (* On va prouver que ce contexte est incohérent. *)
```

```
  apply H'.
```

```
  exact H.
```

```
Qed.
```

Évidemment, n'utiliser `exfalso` que lorsqu'on est sûr que le contexte est contradictoire, et revenir en arrière si on s'aperçoit que ce n'est pas le cas.

La négation

- ▶ En Coq (et dans beaucoup de logiques), la négation d'une proposition A est simplement une notation pour $A \rightarrow \text{False}$.
- ▶ On la note en Coq $\sim A$ (et cela peut apparaître comme $\neg A$ suivant l'interface utilisée).
- ▶ On *peut* utiliser la tactique **unfold** pour déplier la notation.

Lemma absurd' : forall (A P : Prop), A -> ~ A -> P.

Proof.

```
unfold not. (* change ~ A en A -> False *)
intros A P H H'.
exfalso. (* On va prouver que ce contexte est incohérent. *)
apply H'.
exact H.
```

Qed.

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Équivalence logique

- ▶ En Coq, l'équivalence logique est définie à partir de l'implication et du « et ».
- ▶ **About** " $\leftrightarrow$ ". (** iff : Prop -> Prop -> Prop **)
Print iff. (** iff = fun (A B : Prop) => (A -> B) /\ (B -> A). **)
- ▶ On a donc juste à utiliser ce qu'on sait déjà sur $\wedge$ et $\rightarrow$.
- ▶ Pour prouver une équivalence, on utilisera donc **split** et on prouvera les deux implications comme on sait le faire (en général avec **intros**).
- ▶ Pour utiliser une équivalence, on utilisera **destruct** H **as** [H1 H2] ou un *intro pattern*, puis des *modus ponens* pour utiliser les deux implications.

Exemple de preuve d'une équivalence

Lemma and_comm : forall (A B : Prop), A /\ B <=> B /\ A.

Proof.

intros A B.

split.

- (* Preuve de $A \wedge B \rightarrow B \wedge A$: *)

intros [H1 H2].

split.

+ (* Preuve de B : *)

exact H2.

+ (* Preuve de A : *)

exact H1.

- (* Preuve de $B \wedge A \rightarrow A \wedge B$: *)

intros [H1 H2].

split.

+ (* Preuve de A : *)

exact H2.

+ (* Preuve de B : *)

exact H1.

Qed.

Exemple d'utilisation d'une équivalence

```
Lemma imp_iff_compat_l : forall (A B C : Prop),  
  (A <-> B) -> ((A -> C) <-> (B -> C)).
```

```
Proof.
```

```
  intros A B C [H1 H2].
```

```
  split.
```

```
  - (* Preuve de (A -> C) -> B -> C : *)
```

```
    intros H H'.
```

```
    apply H.
```

```
    apply H2.
```

```
    exact H'.
```

```
  - (* Preuve de (B -> C) -> A -> C : *)
```

```
    intros H H'.
```

```
    apply H.
```

```
    apply H1.
```

```
    exact H'.
```

```
Qed.
```

Organisation du module

Les assistants de preuve et Coq/Rocq

Déduction naturelle en Coq

Introduction du « pour tout »

Règles de l'implication

Règles de la conjonction (« et »)

Règles de la disjonction (« ou »)

Le « faux » et la négation

L'équivalence

Résumé des tactiques de déduction naturelle en Coq

Tactiques et règles de déduction naturelle

On a déjà passé en revue presque toutes les règles de déduction naturelle en Coq. Les règles du quantificateur existentiel $\exists$ et la règle d'élimination de $\forall$ seront vues plus tard.

connecteur	introduction (prouver)	élimination (utiliser)
$\rightarrow$	intros	apply
$\wedge$	split	destruct
$\vee$	left ou right	destruct
$\forall$	intros	specialize , instanciation ¹
$\perp$		destruct
$\exists$	exists	destruct

1. explicite ou implicite avec unification