

Déduction naturelle et tactiques en Lean 4

Marie-Hélène Mourgues, Antoine Meyer

March 18, 2025

1 Introduction

Démonstration vs. utilisation. Dans la présentation ci-dessous, on distingue des règles qui permettent d'établir un nouveau fait ayant un certain connecteur principal (règles de « démonstration »), et des règles qui permettent d'utiliser des hypothèses ou des théorèmes ayant un certain connecteur principal (règles d'« utilisation »).

Dans certains contextes (par exemple en Lean) on dit aussi « introduction » à la place de démonstration, et « élimination » à la place d'utilisation, d'où le « i » et le « e » dans le nom abrégé des règles, ou le nom du théorème `And.intro` en Lean par exemple.

La notation \vdash . Dans les règles qui suivent, on utilise la notation $\mathcal{H} \vdash P$ pour exprimer le fait que sous la liste d'hypothèses \mathcal{H} , il est possible de démontrer l'assertion P (ce qui peut être vrai ou non !). Le symbole \vdash est appelé *turnstile* (tourniquet) en anglais. Cette notation est également utilisée en Lean pour séparer la liste des objets et hypothèses courants de l'énoncé à démontrer.

On appellera parfois \mathcal{H} le *contexte* ou les *prémisses*, et P le *but*. Le couple $\mathcal{H} \vdash P$ est parfois qualifié d'*état* de la preuve.

Une *preuve formelle* consiste à combiner et arranger entre elles de manière très rigoureuse des affirmations de ce type (des états donc) à l'aide d'un système de règles, jusqu'à être en mesure de démontrer que l'énoncé qu'on a considéré au début (le but initial) est bel et bien une conséquence des axiomes et des règles de raisonnement que l'on s'est choisi, dans le contexte considéré.

2 Règles de la déduction naturelle

2.1 Règle sans prémisses

On utilise la règle suivante pour « clore » une démonstration quand le fait à démontrer est déjà connu (c'est à dire quand il apparaît déjà dans le contexte) :

$$\frac{}{\mathcal{H} \cup \{P\} \vdash P} \text{Ax}$$

On voit qu'il n'y a pas de prémisses (rien au-dessus de la barre horizontale), ce qui signifie qu'il n'y a « rien d'autre à faire » pour établir la véracité de P . Dans une démonstration en français, on pourrait écrire « par hypothèse ».

2.2 Le connecteur \Rightarrow (implique)

2.2.1 démonstrations

$$\frac{\mathcal{H} \cup \{P\} \vdash Q}{\mathcal{H} \vdash P \Rightarrow Q} \Rightarrow_i$$

Cette règle exprime le fait que pour démontrer $P \Rightarrow Q$, il suffit de démontrer Q dans le contexte de départ auquel on a ajouté l'hypothèse P . Ceci correspond à un raisonnement du type « Supposons P , et montrons qu'on a alors Q . ». Attention, dans la conclusion (ligne inférieure de la règle) P n'est plus dans les hypothèses, on a seulement supposé P le temps de la démonstration de Q .

2.2.2 Utilisation

$$\frac{\mathcal{H} \vdash P \quad \mathcal{H} \vdash P \Rightarrow Q}{\mathcal{H} \vdash Q} \Rightarrow_e$$

Si dans le contexte courant on est capable de démontrer $P \Rightarrow Q$ d'une part, et P d'autre part, alors on peut aussi démontrer Q . Cette règle bien connue est également appelée *modus ponens*.

2.3 La conjonction \wedge (« et »)

2.3.1 Démonstration

$$\frac{\mathcal{H} \vdash P \quad \mathcal{H} \vdash Q}{\mathcal{H} \vdash P \wedge Q} \wedge_i$$

Cette règle signifie que pour démontrer une proposition du type $P \wedge Q$ sous la liste d'hypothèses \mathcal{H} (ligne du bas), il faut pouvoir séparément démontrer P et démontrer Q avec les mêmes hypothèses (deux parties de la ligne du haut).

Exemple. Pour démontrer $-1 \leq x < 3$ qui est en fait $-1 \leq x$ et $x < 3$ on doit démontrer $-1 \leq x$ et $x < 3$.

2.3.2 Utilisation

Il existe deux variantes de la règle, une à gauche (\wedge_{eg}) et une à droite (\wedge_{ed}).

$$\frac{\mathcal{H} \vdash P \wedge Q}{\mathcal{H} \vdash P} \wedge_{eg} \quad \frac{\mathcal{H} \vdash P \wedge Q}{\mathcal{H} \vdash Q} \wedge_{ed}$$

Ces règles signifient que si l'on parvient à démontrer $P \wedge Q$, alors on peut en déduire une preuve de P et une preuve de Q .

2.4 La disjonction \vee (« ou »)

2.4.1 Démonstration

Il y a deux manières de démontrer $P \vee Q$: prouver P , ou bien prouver Q .

$$\frac{\mathcal{H} \vdash P}{\mathcal{H} \vdash P \vee Q} \vee_{ig} \quad \frac{\mathcal{H} \vdash Q}{\mathcal{H} \vdash P \vee Q} \vee_{id}$$

Choisir l'une ou l'autre de ces options ne dit bien sûr rien sur la véracité de celle qui n'a pas été choisie...

Exemple : si A et B sont deux sous-ensembles d'un ensemble E et x est un élément de E , pour prouver que $x \in A \cup B$, c'est à dire prouver que $x \in A$ ou $x \in B$, il suffit de prouver que $x \in A$ (ou que $x \in B$). Bien sûr il peut arriver que les deux propriétés soient vraies, mais dans ce cas précis cela ne sert à rien de le savoir.

2.4.2 Utilisation

$$\frac{\mathcal{H} \vdash P \vee Q \quad \mathcal{H} \vdash P \Rightarrow R \quad \mathcal{H} \vdash Q \Rightarrow R}{\mathcal{H} \vdash R} \vee_e$$

Si dans le contexte actuel on peut démontrer $P \vee Q$, et qu'on peut par ailleurs démontrer R quelle que soit la raison pour laquelle $P \vee Q$ est vraie (c'est à dire d'une part en supposant P et *indépendamment* en supposant Q), alors on peut en déduire que R est vraie. C'est ce que l'on appelle le raisonnement *par disjonction des cas*.

On utilise souvent ce raisonnement pour démontrer une proposition universelle de la forme $\forall x \in E P(x)$ avec $E = E_1 \cup E_2$. Si on sait que $x \in E$, alors soit $x \in E_1$, soit $x \in E_2$. D'après la règle d'élimination du \vee , il suffit donc de montrer que si $x \in E_1$ alors $P(x)$ et si $x \in E_2$ alors $P(x)$.

Exemple : on veut montrer que pour tout entier n , $n(n+1)/2$ est un entier. On sait que tout entier n , n est pair ou n est impair. Si n est pair alors $n/2$ est un entier donc $n(n+1)/2$ est un entier. Si n est impair alors $n+1$ est pair et donc $(n+1)/2$ est un entier donc $n(n+1)/2$ est un entier.

2.5 L'équivalence \Leftrightarrow (« si et seulement si »)

L'équivalence propositionnelle \Leftrightarrow , souvent lue « si et seulement si », peut être vue comme une simple abréviation d'une conjonction d'implications :

$$P \Leftrightarrow Q \text{ signifie } (P \Rightarrow Q) \wedge (Q \Rightarrow P)$$

Ainsi, il n'est pas indispensable d'introduire de nouvelles règles pour cet opérateur, et d'utiliser à la place les règles de la conjonction. Cependant, pour obtenir des démonstrations plus concises, on peut introduire des règles spécifiques (on parle de *règles dérivées* puisqu'il est possible de les déduire des autres règles).

2.5.1 Démonstration

$$\frac{\mathcal{H} \vdash P \Rightarrow Q \quad \mathcal{H} \vdash Q \Rightarrow P}{\mathcal{H} \vdash P \Leftrightarrow Q} \Leftrightarrow_i$$

Cette règle signifie que pour démontrer une proposition du type $P \Leftrightarrow Q$ il faut pouvoir séparément démontrer l'implication directe et l'implication réciproque. Ce principe familier est souvent appelé *preuve par double implication*.

2.5.2 Utilisation

On utilise ici deux règles, l'une pour l'implication directe (\Leftrightarrow_{ed}) et l'autre pour l'implication réciproque (\Leftrightarrow_{er}).

$$\frac{\mathcal{H} \vdash P \Leftrightarrow Q}{\mathcal{H} \vdash P \Rightarrow Q} \Leftrightarrow_{ed} \quad \frac{\mathcal{H} \vdash P \Leftrightarrow Q}{\mathcal{H} \vdash Q \Rightarrow P} \Leftrightarrow_{er}$$

2.6 La négation \neg (« non »)

2.6.1 Démonstration

Pour démontrer la négation d'un énoncé, il suffit de montrer qu'ajouter cet énoncé au contexte permet de dériver deux énoncés contradictoires.

$$\frac{\mathcal{H} \cup \{P\} \vdash Q \quad \mathcal{H} \cup \{P\} \vdash \neg Q}{\mathcal{H} \vdash \neg P} \neg_i$$

En d'autres termes, si l'ajout de l'hypothèse P au contexte conduit à une contradiction (c'est à dire que l'on sait démontrer à la fois Q et $\neg Q$ pour un certain énoncé Q), alors peut en déduire $\neg P$.

2.6.2 Utilisation

Si l'on souhaite démontrer la double négation d'un énoncé, il est suffisant de démontrer l'énoncé lui-même :

$$\frac{\mathcal{H} \vdash \neg \neg P}{\mathcal{H} \vdash P} \neg_e$$

Cette règle porte aussi le nom de *règle d'introduction de la double négation*.

Les deux règles précédentes peuvent être combinées pour effectuer un raisonnement par l'absurde (appelé aussi raisonnement par *contradiction* en anglais). Pour montrer P , on peut supposer (temporairement) $\neg P$. Si l'on aboutit dans ce nouveau contexte à une contradiction, on en déduit $\neg \neg P$ par la règle de démonstration, puis P par la règle d'utilisation.

2.7 Le quantificateur universel \forall (« pour tout »)

2.7.1 Démonstration

$$\frac{\mathcal{H} \vdash P(x)}{\mathcal{H} \vdash \forall x P(x)} \forall_e \quad \text{avec } x \text{ non libre dans } \mathcal{H}$$

Cette règle un peu étrange signifie exactement que, pour que $\forall x P(x)$ soit vraie, il suffit que $P(x)$ soit vraie pour un élément x est quelconque, c'est à dire un élément x sur lequel on n'a fait aucune hypothèse. Cette règle ne s'applique donc que si x n'apparaît librement dans aucune des hypothèses du contexte \mathcal{H} (sinon on choisit simplement une autre lettre).

C'est ainsi que l'on procède en pratique: lorsque l'on veut démontrer une proposition du type $\forall x \in E, P(x)$, on commence la démonstration par « soit x », où x est une variable « fraîche » désignant un élément quelconque de l'ensemble E dans lequel on travaille.

2.7.2 Utilisation

Si l'on sait démontrer qu'une propriété est vraie pour tout élément de l'ensemble de travail, alors on peut en déduire que cette propriété est vraie de n'importe quel élément particulier de l'ensemble :

$$\frac{\mathcal{H} \vdash \forall x P(x)}{\mathcal{H} \vdash P(a)} \forall_e$$

Dans cette règle, a est n'importe quel élément de l'ensemble dans lequel on travaille. On dit qu'on a *instancié* la propriété $\forall x P(x)$ à l'élément a .

2.8 Le quantificateur existentiel \exists (« il existe »)

2.8.1 Démonstration

La règle de démonstration d'une propriété existentielle $\exists x P(x)$ est

$$\frac{\mathcal{H} \vdash P(a)}{\mathcal{H} \vdash \exists x P(x)} \exists_i$$

où a est un élément de l'ensemble dans lequel on travaille.

Pour démontrer une proposition du type $\exists x P(x)$, on doit généralement trouver (*exhiber*) un élément pour lequel cette propriété est vraie. On appelle aussi a un *témoin* pour la propriété $\exists x P(x)$.

On note que cette propriété n'affirme pas qu'il existe un seul témoin, mais simplement qu'il en existe au moins un.

2.8.2 Utilisation

La règle d'utilisation d'une propriété existentielle est

$$\frac{\mathcal{H} \vdash \exists x P(x) \quad \mathcal{H} \cup \{P(a)\} \vdash Q}{\mathcal{H} \vdash Q} \exists_e$$

où a est un élément quelconque de l'ensemble où l'on travaille, c'est à dire que a n'apparaît pas comme variable libre dans le contexte.

Attention, il peut être trompeur de réutiliser la lettre x pour nommer le témoin ! Dans une démonstration rédigée en français, on peut par exemple pour éviter toute erreur écrire « de $\exists x P(x)$, on obtient un certain a tel que $P(a)$ », ou encore « par hypothèse, il existe un élément sur lequel P est vraie, soit a un tel élément ».

2.9 Récapitulatif des règles

La figure 1 récapitule la plupart des règles de démonstration et d'utilisation (aussi appelées règles d'introduction et d'élimination) mentionnées ci-dessus.

Op.	Démonstration	Utilisation
\Rightarrow	$\frac{\mathcal{H} \cup \{P\} \vdash Q}{\mathcal{H} \vdash P \Rightarrow Q} \Rightarrow_i$	$\frac{\mathcal{H} \vdash P \Rightarrow Q \quad \mathcal{H} \vdash P}{\mathcal{H} \vdash Q} \Rightarrow_e$
\neg	$\frac{\mathcal{H} \cup \{P\} \vdash Q \quad \mathcal{H} \cup \{P\} \vdash \neg Q}{\mathcal{H} \vdash \neg P} \neg_i$	$\frac{\mathcal{H} \vdash \neg \neg P}{\mathcal{H} \vdash P} \neg_e$
\vee	$\frac{\mathcal{H} \vdash P}{\mathcal{H} \vdash P \vee Q} \vee_{ig} \quad \frac{\mathcal{H} \vdash Q}{\mathcal{H} \vdash P \vee Q} \vee_{id}$	$\frac{\mathcal{H} \vdash P \vee Q \quad \mathcal{H} \cup \{P\} \vdash R \quad \mathcal{H} \cup \{Q\} \vdash R}{\mathcal{H} \vdash R} \vee_e$
\wedge	$\frac{\mathcal{H} \vdash P \quad \mathcal{H} \vdash Q}{\mathcal{H} \vdash P \wedge Q} \wedge_i$	$\frac{\mathcal{H} \vdash P \wedge Q}{\mathcal{H} \vdash P} \wedge_{eg} \quad \frac{\mathcal{H} \vdash P \wedge Q}{\mathcal{H} \vdash Q} \wedge_{ed}$
\forall	$\frac{\mathcal{H} \vdash P(x)}{\mathcal{H} \vdash \forall x P(x)} \forall_e$ (x sans occurrence libre dans \mathcal{H})	$\frac{\mathcal{H} \vdash \forall x P(x)}{\mathcal{H} \vdash P(a)} \forall_e$ (a objet quelconque du domaine)
\exists	$\frac{\mathcal{H} \vdash P(a)}{\mathcal{H} \vdash \exists x P(x)} \exists_i$	$\frac{\mathcal{H} \vdash \exists x P(x) \quad \mathcal{H} \cup \{P(a)\} \vdash Q}{\mathcal{H} \vdash Q} \exists_e$

Figure 1: Principales règles de déduction naturelle

3 Analogues en Lean

3.1 Règle sans prémisses

Quand le contexte contient une hypothèse `h : p` et que le but courant est `p` (ou un énoncé équivalent à `p` en un certain sens), on peut clore le but courant grâce à la tactique `exact h`. Dans certains cas (comme on le verra par exemple dans la section 2.2.2) il peut être utile de combiner entre eux plusieurs objets et hypothèses du contexte.

D'autres tactiques plus puissantes comme `assumption` ou `trivial` recherchent automatiquement dans le contexte des hypothèses permettant de clore immédiatement le but courant.

3.2 Le connecteur \Rightarrow (implique)

3.2.1 Démonstration

En Lean 4, la flèche d'implication est notée \rightarrow (taper `\r` ou `\imp`). Pour démontrer un but de la forme `p \rightarrow q`, on commence typiquement par utiliser la tactique `intro h`, avec `h` un nouvel identifiant d'hypothèse. Son effet est d'ajouter au contexte l'hypothèse `h : p` et de transformer le but courant en `q`. Ce mécanisme est tout à fait analogue à la règle \Rightarrow_i ci-dessus.

3.2.2 Utilisation

Si `h : p \rightarrow q` est déjà dans le contexte, ou s'il existe dans l'espace de noms actuel un théorème semblable, et que le but courant est `q` (ou une expression équivalente à `q`), alors la tactique `apply h` transforme le but courant en `p`. Cela signifie en quelque sorte que pour démontrer `q`, il *suffit* de démontrer `p` car `h : p \rightarrow q`. Ce mécanisme est une traduction en Lean du principe de *modus ponens*.

Si l'on dispose d'une hypothèse `hp : p`, on peut également *spécialiser* `h` à l'aide de la tactique `specialize h hp`. L'hypothèse `h : p \rightarrow q` est alors remplacée par `h : q`.

3.3 La conjonction \wedge (« et »)

3.3.1 Démonstration

Pour démontrer un but de la forme `p \wedge q`, il existe plusieurs possibilités :

- Tactique `constructor`.
- Utilisation du théorème `And.intro` (par exemple à l'aide de la tactique `apply`, en écrivant `apply And.intro`).

Chacune de ces deux méthodes génère deux buts, un pour `p` et un pour `q`. On peut utiliser un point (`.`) ou un point médian (`.|.`) pour structurer la suite de la preuve et traiter chaque but séparément.

On peut aussi appliquer directement `And.intro` à des expressions calculant des preuves de `p` et `q`. Par exemple si on a dans le contexte `hp : p` et `hq : q` on peut clore le but directement avec `exact And.intro hp hq`.

3.3.2 Utilisation

Pour démontrer l'énoncé `p`, il est possible de se ramener à démontrer une propriété plus forte de la forme `p ∧ q` grâce à la fonction d'élimination `And.left` (ou de la forme `q ∧ p` avec `And.right`). Par exemple depuis l'état

```
p : Prop
⊢ p
```

appliquer la tactique `apply And.left` amène à l'état

```
p : Prop
⊢ p ∧ ?b
```

où `?b` désigne une propriété quelconque.

Dans le cas où le contexte contient déjà une hypothèse `h : p ∧ q` :

```
p q : Prop
h : p ∧ q
⊢ p
```

on peut directement en déduire une preuve de `p` en écrivant

```
exact And.left h
```

ou de manière plus concise `exact h.left` (de même, on pourrait déduire une preuve de `q` par `exact h.right`). Alternativement, si l'état de preuve est

```
p q : Prop
h : p ∧ q
⊢ r
```

on peut « découper » `h` en deux nouvelles hypothèses `hp : p` et `hq : q` grâce à la tactique

```
rcases h with <hp, hq>
```

(attention, ce ne sont pas ici des parenthèses mais des angles, saisis à l'aide des commandes `\<` et `\>`), ce qui amène à l'état de preuve

```
p q : Prop
hp : p
hq : q
├ r
```

3.4 La disjonction \vee (« ou »)

3.4.1 Démonstration

L'analogue Lean de la règle \vee_{ig} est la tactique `left` ou, de manière quasiment équivalente, `apply Or.inl`. La règle \vee_{id} correspond quant à elle aux tactiques `right` ou `apply Or.inr`. Ces tactiques sont utilisables quand le but est de la forme `p \vee q`, et remplacent respectivement le but courant par `p` et par `q`.

3.4.2 Utilisation

Si l'on suppose que le contexte courant contient l'hypothèse `h : p \vee q` et que le but courant est `r`, on peut créer deux nouveaux états de preuve avec la tactique `cases h`. Dans le premier, on dispose d'une nouvelle hypothèse de type `p`, et dans le deuxième d'une nouvelle hypothèse de type `q`. Dans les deux cas, il faut parvenir à démontrer le but initial `r`. Pour pouvoir donner un nom aux nouvelles hypothèses, on peut utiliser la syntaxe plus complète suivante :

```
cases h with
| Or.inl hg => -- preuve du but en supposant (hg : p)
| Or.inr hd => -- preuve du but en supposant (hd : q)
```

Une autre syntaxe possible (essentiellement équivalente dans ce cas) est :

```
rcases h with (hg | hd)
```

3.5 L'équivalence \Leftrightarrow (« si et seulement si »)

3.5.1 Démonstration

Le symbole d'équivalence s'écrit \Leftrightarrow en Lean (saisir `\iff`). La tactique la plus simple pour démontrer une équivalence est `constructor`, comme dans le cas de la conjonction. Comme précédemment, si le but est de la forme `p \Leftrightarrow q`, appliquer cette tactique génère deux buts, un pour démontrer `p \rightarrow q` et l'autre pour `q \rightarrow p`.

3.5.2 Utilisation

Dans le cas où le contexte contient déjà une hypothèse `h : p ↔ q` :

```
p q : Prop
h : p ↔ q
⊢ p → q
```

on peut directement en déduire une preuve de `p → q` en écrivant

```
exact Iff.mp h
```

ou de manière plus concise `exact h.mp` (de même, on pourrait déduire une preuve de `q → p` par `exact h.mpr`)¹. Si l'état de preuve est

```
p q : Prop
h : p ↔ q
⊢ r
```

pour un certain but `r`, on peut aussi « découper » `h` en deux nouvelles hypothèses `hpq : p → q` et `hqp : q → p` grâce à la tactique

```
rcases h with <hp, hq>
```

(attention, ce ne sont pas ici des parenthèses mais des angles, saisis à l'aide des commandes `\<` et `\>`), ce qui amène à l'état de preuve

```
p q : Prop
hpq : p → q
hqp : q → p
⊢ r
```

Enfin, dans l'état de preuve

```
p q : Prop
h : ∀ x, p x ↔ q x
⊢ p a
```

on peut directement « réécrire » `p` en `q` dans le but en appliquant la tactique `rewrite [h]`, ce qui a pour effet de remplacer le but courant par `q a`. Réciproquement, si le but est de la forme `q a` on peut le remplacer par `p a` en appliquant la tactique `rewrite [<h]` (taper `\l` pour obtenir la flèche vers la gauche).

¹Les noms `mp` et `mpr` sont censés évoquer *modus ponens* et *modus ponens réciproque*.

3.6 La négation \neg (« non »)

3.6.1 Démonstration

En Lean, un énoncé de la forme `$\neg p$` est en réalité équivalent à `$p \rightarrow \text{False}$` , où `False` est une proposition spéciale pour laquelle il ne peut pas exister de preuve. Par conséquent, quand le but courant est de la forme `$\neg p$` on peut appliquer la tactique `intro h`, qui ajoute au contexte l'hypothèse `$h : p$` et transforme le but courant en `False`.

3.6.2 Utilisation

Réciproquement, quand le but est `False` et qu'on dispose d'une hypothèse de la forme `$h : \neg p$` , on peut utiliser la tactique `apply h` pour changer le but en `p` . Ce mécanisme est un peu contre-intuitif, et demande de la pratique afin de s'y habituer.

Comment démontrer `False` ? Comme on l'a dit précédemment, il ne peut exister de preuve directe de `False`. Pour clore un but de type `False`, les seules possibilités sont donc :

- disposer d'une hypothèse `$h : \text{False}$` , auquel cas on peut clore le but avec `exact h` ;
- disposer d'une hypothèse de type `$h : \neg q$` , auquel cas on peut changer le but en `q` avec `apply h` ;
- montrer qu'il existe deux hypothèses contradictoires dans le contexte (par exemple de types `q` et `$\neg q$`), ou une hypothèse « impossible » (par exemple `$0 = 1$`), auquel cas on peut clore le but courant par la tactique `contradiction`.

3.7 Le quantificateur universel \forall (« pour tout »)

3.7.1 Démonstration

La syntaxe permettant d'énoncer la propriété universelle $\forall x P(x)$ en Lean est `$\forall x, e$` (saisir `\all` pour obtenir le symbole `\forall`) où `e` est une expression faisant éventuellement intervenir `x`.

Dans la théorie sous-jacente de Lean, il n'existe en réalité pas de grande différence entre propriétés universelles et implications. Par conséquent, les tactiques et syntaxes qui s'appliquent au premier type de propriété s'appliquent aussi au second, à la fois quand on veut démontrer un but ou quand on veut utiliser une hypothèse.

En particulier, si le but courant est de la forme `$\forall x, e$` , on peut ajouter un nouvel objet au contexte grâce à la tactique `intro`, comme pour un but

contenant une implication. On peut nommer ce nouvel objet en précisant un identifiant, par exemple `intro x` ou `intro y`. Si l'identifiant choisi existe déjà dans le contexte, il est alors *masqué* par le nouvel identifiant.

3.7.2 Utilisation

Comme indiqué ci-dessus, l'utilisation d'une hypothèse ou d'un théorème de la forme `h : ∀ x, e` où `e` est une expression dans laquelle `x` peut apparaître comme variable libre, est similaire à l'utilisation d'une hypothèse de type `p → q`.

En particulier, on peut construire une instanciation de `h` à un objet `a` du type de `x` en écrivant `h a`. Cette nouvelle expression est de type `e'`, où `e'` est obtenu en remplaçant toutes les occurrences libres de `x` par `a` (on dit qu'on *substitue* `a` à `x` dans `e`).

Si le but courant est de la forme `e'`, on peut directement utiliser la tactique `apply h` pour clore le but. Lean peut généralement inférer automatiquement quelle expression il convient d'utiliser pour `a`.

Enfin, on peut instancier directement une hypothèse `h : ∀ x, e` du contexte en écrivant par exemple `specialize h a` ou `replace h := h a`. Cela a pour effet de remplacer `h : ∀ x, e` par `h : e'` dans le contexte.

3.8 Le quantificateur existentiel \exists (« il existe »)

3.8.1 Démonstration

La tactique la plus couramment employée pour démontrer un but de la forme `∃ x, e` est `exists a`, qui a pour effet de remplacer le but par `e'`, où `e'` est obtenue en substituant `a` à `x` dans `e`. Comme dans la plupart des cas, `a` peut être une variable du contexte ou une expression plus complexe. Dans les cas particulièrement simples, il peut arriver que le but `e'` soit clos automatiquement, par exemple si c'est une égalité triviale.

3.8.2 Utilisation

Si le contexte contient une hypothèse `h : ∃ x, e`, la tactique

```
obtain <a, ha> := h
```

permet d'introduire un nouvel objet `a` et une hypothèse `ha : e'` où `e'` est obtenue en substituant `x` par `a` dans `e` (attention, comme pour la tactique `rcases`, ce ne sont pas ici des parenthèses mais des angles, saisis à l'aide des commandes `\<` et `\>`).

Op.	État initial	Tactique	État final
\Rightarrow	<code>⊢ p → q</code>	<code>intro h</code>	<code>h : p</code> <code>⊢ q</code>
\neg	<code>⊢ ¬ p</code>	<code>intro h</code>	<code>h : p</code> <code>⊢ False</code>
\wedge	<code>⊢ p ∧ q</code>	<code>constructor</code>	<code>⊢ p</code> et <code>⊢ q</code>
\vee	<code>⊢ p ∨ q</code> <code>⊢ p ∨ q</code>	<code>left</code> <code>right</code>	<code>⊢ p</code> <code>⊢ q</code>
\forall	<code>⊢ ∀ x, P x</code>	<code>intro a</code>	<code>a : ...</code> <code>⊢ P a</code>
\exists	<code>⊢ ∃ x, P x</code>	<code>exists a</code>	<code>⊢ P a</code>

Figure 2: Quelques tactiques Lean utiles, par opérateur (démontrer / introduire)

3.9 Récapitulatif des tactiques par opérateur


Les figures 2 et 3 récapitulent les principales tactiques Lean utilisées pour démontrer ou utiliser les propriétés contenant chacun des opérateurs logiques.

Il est utile de comparer l'effet de ces tactiques sur l'état de la preuve aux règles de déduction naturelle rassemblées dans la Fig. 1. Cependant, on gardera en tête que ces tactiques ne correspondent pas toujours *exactement* aux règles de déduction naturelle, mais doivent plutôt être vues comme des *analogues*. En particulier, certaines d'entre elles transforment le contexte et non le but, ou correspondent à une composition de plusieurs règles de déduction naturelle.

4 Types et termes de preuve en Lean

Dans la théorie sous-jacente à Lean (correspondance entre preuves et programmes et entre propositions et types), une implication est vue comme un type fonctionnel, et la preuve d'une implication comme une fonction pouvant s'appliquer à une preuve de sa prémisse. Ainsi, si l'on est capable dans le contexte actuel d'écrire une expression `e : p` (par exemple si on a directement une hypothèse de ce type), on peut aussi immédiatement clore le but en écrivant `exact h e`.

5 Liste de tactiques Lean utiles

 *Section en chantier !*

`apply`

`assumption`

Op.	État initial	Tactique	État final
\Rightarrow	$\begin{array}{l} h : p \rightarrow q \\ \vdash q \end{array}$	<code>apply h</code>	$\begin{array}{l} h : p \rightarrow q \\ \vdash p \end{array}$
\neg	$\begin{array}{l} h : \neg p \\ \vdash \text{False} \end{array}$	<code>apply h</code>	$\begin{array}{l} h : \neg p \\ \vdash p \end{array}$
\wedge	$\begin{array}{l} h : p \wedge q \\ \vdash r \end{array}$	<code>rcases h</code> <code> with <hp, hq></code>	$\begin{array}{l} hp : p \\ hq : q \\ \vdash r \end{array}$
\vee	$\begin{array}{l} h : p \vee q \\ \vdash r \end{array}$	<code>rcases h</code> <code> with (hp hq)</code>	$\begin{array}{l} hp : p \\ \vdash r \end{array}$ et $\begin{array}{l} hq : q \\ \vdash r \end{array}$
\forall	$\begin{array}{l} h : \forall x, P x \\ \vdash P a \end{array}$	<code>apply h</code>	but clos
	$\begin{array}{l} h : \forall x, P x \\ \vdash q \end{array}$	<code>specialize h a</code>	$\begin{array}{l} h : P a \\ \vdash q \end{array}$
\exists	$\begin{array}{l} h : \exists x, P x \\ \vdash q \end{array}$	<code>obtain <a, ha> := h</code>	$\begin{array}{l} a : \dots \\ ha : P a \\ \vdash q \end{array}$

Figure 3: Quelques tactiques Lean utiles, par opérateur (utiliser / éliminer)

`constructor`
`contradiction`
`exfalse`
`exists`
`have`
`induction`
`intro`
`left` et `right`
`obtain`
`rcases`
`rewrite`
`rfl`
`simp`
`sorry`
`specialize`
`trivial`

6 Exemples de démonstrations

6.1 Arbres de démonstration en déduction naturelle

Nous allons montrer en utilisant une suite de règles qu'une implication est équivalente à sa contraposée: $(P \Rightarrow Q) \Leftrightarrow (\neg Q \Rightarrow \neg P)$.

Pour simplifier, nous allons donner un arbre pour la démonstration de $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$ et un arbre pour la démonstration de la réciproque $(\neg Q \Rightarrow \neg P) \Rightarrow (P \Rightarrow Q)$, il suffira ensuite de la règle de démonstration du \Leftrightarrow pour conclure. Nous n'avons besoin d'aucune hypothèse ici donc \mathcal{H} sera l'ensemble vide et nous l'omettrons dans l'arbre.

L'arbre de preuve de $((P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P))$ est donné Fig. 4, celui de $(\neg Q \Rightarrow \neg P) \Rightarrow (P \Rightarrow Q)$ Fig. 5.

6.2 Démonstrations Lean

Les figures 6 et 7 détaillent des preuves analogues en Lean. Malgré des différences, on retrouve la même idée générale de la preuve.

$$\begin{array}{c}
\frac{\frac{\text{Ax}}{\{P, P \Rightarrow Q, \neg Q\} \vdash P} \quad \frac{\text{Ax}}{\{P, P \Rightarrow Q, \neg Q\} \vdash P \Rightarrow Q}}{\{P, P \Rightarrow Q, \neg Q\} \vdash Q} \Rightarrow_e \quad \frac{\text{Ax}}{\{P, P \Rightarrow Q, \neg Q\} \vdash \neg Q} \neg_i \\
\hline
\frac{\{P \Rightarrow Q, \neg Q\} \vdash \neg P}{\{P \Rightarrow Q\} \vdash \neg Q \Rightarrow \neg P} \Rightarrow_i \\
\hline
\frac{}{\vdash (P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)} \Rightarrow_i
\end{array}$$

Figure 4: Démonstration de $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$.

$$\begin{array}{c}
\frac{\frac{\text{Ax}}{\{\neg Q \Rightarrow \neg P, P, \neg Q\} \vdash P} \quad \frac{\text{Ax}}{\{\neg Q \Rightarrow \neg P, P, \neg Q\} \vdash \neg Q \Rightarrow \neg P} \quad \frac{\text{Ax}}{\{\neg Q \Rightarrow \neg P, P, \neg Q\} \vdash \neg Q} \Rightarrow_e}{\{\neg Q \Rightarrow \neg P, P\} \vdash \neg \neg Q} \neg_e \\
\frac{\{\neg Q \Rightarrow \neg P, P\} \vdash Q}{\{\neg Q \Rightarrow \neg P\} \vdash P \Rightarrow Q} \Rightarrow_i \\
\hline
\frac{}{\vdash (\neg Q \Rightarrow \neg P) \Rightarrow (P \Rightarrow Q)} \Rightarrow_i
\end{array}$$

Figure 5: Démonstration de $(\neg Q \Rightarrow \neg P) \Rightarrow (P \Rightarrow Q)$.

Commentaires et tactiques	État de la preuve
Soient <code>p</code> et <code>q</code> deux propositions. On suppose <code>p → q</code> .	<code>p q : Prop</code> <code>⊢ (p → q) → ¬q → ¬p</code>
<code>intro hpq</code>	
Montrons <code>¬q → ¬p</code> . On suppose <code>¬q</code> .	<code>p q : Prop</code> <code>hpq : p → q</code> <code>⊢ ¬q → ¬p</code>
<code>intro hnq</code>	
Il reste à prouver <code>¬p</code> , c'est à dire <code>p → False</code> . Supposons <code>p</code> (vers une contradiction).	<code>p q : Prop</code> <code>hpq : p → q</code> <code>hnq : ¬q</code> <code>⊢ ¬p</code>
<code>intro hp</code>	
Il reste à prouver <code>False</code> . De <code>p → q</code> et <code>p</code> on déduit <code>q</code> .	<code>p q : Prop</code> <code>hpq : p → q</code> <code>hnq : ¬q</code> <code>hp : p</code> <code>⊢ False</code>
<code>have hq := hpq hp</code>	
Les hypothèses <code>q</code> et <code>¬q</code> sont contradictoires, on peut en déduire n'importe quel but (notamment <code>False</code>).	<code>p q : Prop</code> <code>hpq : p → q</code> <code>hnq : ¬q</code> <code>hp : p</code> <code>hq : q</code> <code>⊢ False</code>
<code>contradiction</code>	
La preuve est finie !	<i>No goals</i>

Figure 6: Preuve Lean de $(p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)$.

Commentaires et tactiques	État de la preuve
Soient <code>p</code> et <code>q</code> deux propositions. On suppose <code>¬q → ¬p</code> .	<pre>p q : Prop ⊢ (¬q → ¬p) → p → q</pre>
<code>intro hnqnp</code>	
Il reste à montrer <code>p → q</code> . On suppose <code>p</code> .	<pre>p q : Prop hnqnp : ¬q → ¬p ⊢ p → q</pre>
<code>intro hp</code>	
Il reste à montrer <code>q</code> , mais on est coincé. On applique la double négation (c'est un théorème prédéfini du module <code>Classical</code>).	<pre>p q : Prop hnqnp : ¬q → ¬p hp : p ⊢ q</pre>
<code>apply Classical.not_not.mp</code>	
Le but a été transformé en <code>¬¬q</code> , qui est équivalent à <code>¬q → False</code> . On suppose <code>¬q</code> (vers une contradiction).	<pre>p q : Prop hnqnp : ¬q → ¬p hp : p ⊢ ¬¬q</pre>
<code>intro hnq</code>	
Des hypothèses <code>¬q → ¬p</code> et <code>¬q</code> on peut déduire la nouvelle hypothèse <code>¬p</code> (par <i>modus ponens</i>).	<pre>p q : Prop hnqnp : ¬q → ¬p hp : p hnq : ¬q ⊢ False</pre>
<code>have hnp := hnqnp hnq</code>	
Les hypothèses <code>p</code> et <code>¬p</code> étant contradictoires, on peut en déduire n'importe quel but (notamment <code>False</code>).	<pre>p q : Prop hnqnp : ¬q → ¬p hp : p hnq : ¬q hnp : ¬p ⊢ False</pre>
<code>contradiction</code>	
La preuve est finie !	<i>No goals</i>

Figure 7: Preuve Lean de $(\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$.