TP 2 - Propositions et prédicats

1 Ecriture de preuves en LEAN

Avant de commencer, prêtons attention à la manière dont sont écrits les théorèmes dans un livre de mathématiques :

Théorème. (Théorème des valeurs intermédiaires). *Soient $a, b \in \mathbb{R}$ avec $a < b$, et soit $f : [a, b] \longrightarrow \mathbb{R}$ une fonction continue. Pour tout $y \in \mathbb{R}$ compris entre $f(a)$ et $f(b)$, il existe $x \in [a, b]$ tel que $f(x) = y$.*

Preuve. Soient $a, b \in \mathbb{R}$, et soit $f : [a, b] \to \mathbb{R}$ comme dans l'énoncé. [...]

Après le mot « Théorème », on écrit ainsi, dans l'ordre :

1. *un **nom** pour le théorème : ici, c'est "théorème des valeurs intermédiaires" (dans les livres, cela peut-être simplement un numéro ; en* LEAN*, ces noms devront être écrits sous forme de texte) ;*
2. *l'**énoncé** du théorème ;*
3. *une **preuve** pour le théorème.*

La syntaxe fondamentale pour démontrer des théorèmes en LEAN *suit la même idée. La voici :*

```
theorem nom_proposition : enonce_proposition := by
{
        -- Arguments de la preuve
}
```

Ici, il faut remplacer `nom_proposition` *par le nom de son choix en fonction de la situation (par exemple, «* `theoreme_des_valeurs_intermediaires` *»), et* `enonce_proposition` *par l'énoncé pertinent, tapé en* LEAN*, par exemple sous forme d'une formule du premier ordre.*

1. Commencez par taper le code suivant :

```
theorem deux_plus_deux_egal_quatre : (2 + 2 = 4) := by
{
        -- Arguments de la preuve
}
```

Ici, on vient de commencer à énoncer un théorème « `deux_plus_deux_egal_quatre` *», qui affirme que «* $2 + 2 = 4$ *». Si vous placez votre curseur entre les accolades, vous devriez voir ceci :*

```
1 goal
⊢ 2 + 2 = 4
```

Cela signifie que LEAN *a un objectif en cours : montrer que* $2 + 2 = 4$*. Pour accomplir cet objectif, il faut indiquer dans la preuve quelques arguments pour y arriver. Ici, c'est très simple : il suffit de demander à* LEAN *de simplifier l'expression.*

2. Remplacez le commentaire dans la preuve par le mot clé « `simp` ». Que se passe-t-il ?

Voici un exemple d'énoncé de théorème, affirmant que pour tous entiers $n \in \mathbb{Z}$*, si* n *est un carré, alors* n *est positif.*

```
theorem theoreme_carre_implique_positif :
(∀ n : Int, ( ∃ m : Int, n = m*m) → n ≥ 0)
        := by
{
        -- Arguments de la preuve
}
```

Remarquez que l'implication $\Rightarrow$ *s'écrit avec la flèche* $\rightarrow$*. Les retours à la ligne n'ont pas d'importance : vous pouvez les utiliser pour structurer votre code.*

3. Tapez le code précédent.

On peut taper le symbole $\forall$ *en écrivant* `\forall` *et le symbole* $\exists$ *en écrivant* `\exists`*. La plupart du temps, le nom du symbole est celui de sa commande LaTeX, que vous pouvez trouver assez facilement sur Internet.*

Notez aussi que les expressions de la forme $\forall x_1, \ldots, x_n \in E$ *et* $\exists x_1, \ldots, x_n \in E$ *se trouvent converties en LEAN en* `∀x1 ... xn : E` *et* `∃x1 ... xn : E`*.*

4. Ecrivez sur une feuille la formule du premier ordre qui correspond à l'énoncé du théorème précédent.

5. Écrivez des formules du premier ordre correspondant aux énoncés suivants, puis tapez en LEAN des théorèmes affirmant les choses suivantes, sans écrire les preuves.

1. Pour toutes fonctions $f, g : \mathbb{N} \to \mathbb{N}$, si $f \circ g$ est surjective, alors f est surjective.
2. Pour toutes fonctions $f, g : \mathbb{N} \to \mathbb{N}$, si $f \circ g$ est injective, alors g est injective.

2 Ecritures de prédicats

LEAN est capable de stocker en mémoire un énoncé sous la forme d'un objet de type `Prop` *(c'est-à-dire « proposition ».)*

6. Qu'affiche le code suivant ?

```
#check (2 + 2 = 4)
#check (∀ x : Int, ( ∃ y : Int, x = y*y) → x ≥ 0)
```

Etant donné un énoncé, on peut ainsi définir un objet de type `Prop` *qui le contient :*

7. Définissez un objet `enonce_carre_implique_positif`, comme suit :

```
def enonce_carre_implique_positif : Prop :=
        (∀ x : Int, ( ∃ y : Int, x = y*y) → x ≥ 0)
```

Une fois cela fait, on peut se servir de l'énoncé `enonce_implique_carre_positif` pour énoncer un théorème, de la façon suivante :

```
def enonce_carre_implique_positif : Prop :=
        (∀ x : Int, ( ∃ y : Int, x = y*y) → x ≥ 0)

theorem theoreme_carre_implique_positif : enonce_carre_implique_positif
        := by
{
        -- Arguments de la preuve
}
```

Le code ci-dessus est parfaitement synonyme du code vu avant la question **4**, *mais le fait de le décomposer en deux étapes peut-être utile pour structurer son code.*

LEAN permet aussi de définir des prédicats. Un prédicat sur un objet de type `E` *peut-être vu comme une fonction* `E` $\to$ `Prop` *: c'est une fonction qui prend un objet de type* `E`*, et qui renvoie une affirmation dépendant de cet objet.*

Par exemple, le code suivant définit le prédicat "La fonction $f : \mathbb{N} \to \mathbb{N}$ *est croissante".*

```
def est_croissante (f : Nat → Nat) : Prop :=
        ∀ x y : Nat, x ≤ y → (f x) ≤ (f y)
```

8. Que donnent les commandes suivantes ?

```
#check est_croissante

def g : Nat → Nat := (fun x => 2*x)

#check (est_croissante g)
```

Quel est le type de l'objet `est_croissante` ? Comparez le code définissant `est_croissante` à la syntaxe vue au début de la section 2 du TP 1 et vérifiez que vous avez bien compris comment on peut déterminer ce type.

9. Définissez un prédicat `est_surjective` portant sur une fonction $f : \mathbb{Z} \to \mathbb{Z}$ et affirmant que f est surjective. De même, définissez un prédicat `est_injective` portant sur une fonction $f : \mathbb{Z} \to \mathbb{Z}$ et affirmant que f est injective.

Pour l'instant, nous avons expliqué comment on peut énoncer une proposition en LEAN, mais nous ne nous sommes pas encore demandé comment faire pour les démontrer. Nous allons commencer à voir cela dans la section suivante.

3 Ecriture de preuves en LEAN

Une fois définie une proposition sous la forme d'un objet `enonce_proposition` *de type* `Prop`*, il devient possible d'essayer de démontrer ladite proposition, en expliquant à la machine commant en construire une preuve. Rappelons la syntaxe :*

```
theorem nom_proposition : enonce_proposition := by
{
        -- Arguments de la preuve
}
```

(on peut remplacer `nom_proposition` *par le nom que l'on souhaite, mais il est mieux d'utiliser un nom un peu suggestif)*

Tant que les arguments ne sont pas complets, LEAN affiche un message d'erreur. Dans la suite, nous allons apprendre comment écrire de tels arguments en utilisant les outils fournis par le langage.

10. Commencez par taper le code suivant :

```
def enonce_deux_plus_deux_quatre : Prop :=
        ∀ x : Nat, (x = 2) → (x + 2 = 4)

theorem deux_plus_deux_quatre : enonce_deux_plus_deux_quatre := by
{
        -- Arguments de la preuve
}
```

Une fois cela fait, remplacez le commentaire par les lignes suivantes :

```
        rw [enonce_deux_plus_deux_quatre]
        intro x
        intro hyp_x
        rw [hyp_x]
```

Prêtez attention à la manière dont évoluent les messages de la colonne de droite après chaque ligne de code.

Pour plus de clarté dans les messages de la colonne de droite, vous pouvez masquer tous les commentaires, sauf l'onglet "Tactic state", qui va contenir les messages qui nous intéressent.

*Lorsque l'on commence à démontrer une proposition, l'onglet affiche un but (*unsolved goal*) qui contient la proposition à démontrer, ainsi qu'un* contexte*, qui contient les objets et les hypothèses déjà introduites dans la preuve.*
Par exemple, si vous placez le curseur à la fin de la ligne `intro hyp_x`*, vous devez voir la chose suivante :*

1 goal

```
x : Nat
hyp_x : x = 2

⊢ 2 + 2 = 4
```

Cela signifie que :

1. On suppose s'être donné un entier naturel `x`, et on fait l'hypothèse `hyp_x` affirmant que `x = 2` : c'est le contexte.
2. On doit montrer que `x + 2 = 4` : c'est l'**objectif** (**goal**, en anglais).

11. Déplacez le curseur *caractère par caractère* dans les lignes que vous avez écrites à la question **9** précédente. Essayez de comprendre ce qui se passe dans les messages de LEAN (comment évolue le contexte ? comment évolue l'objectif ?)

Dans la question précédente, vous devriez voir que la toute dernière étape consiste à montrer que $2 + 2 = 4$. *En fait,* LEAN *est capable de conclure automatiquement à cette étape, mais il faut parfois lui demander explicitement.*

12. Essayez de taper le code suivant. Déplacez le curseur caractère par caractère dans le code, et regardez ce qui se passe.

```
def enonce_deux_fois_deux_quatre : Prop := (2 * 2 = 4)

theorem deux_fois_deux_quatre : enonce_deux_fois_deux_quatre := by
{
        rw [enonce_deux_fois_deux_quatre]
}

theorem deux_fois_deux_quatre_bis : (2 * 2 = 4) := by
{
        simp
}
```

Comme on va le voir, `rw[enonce_deux_fois_deux_quatre]` *est une commande demandant de réécrire l'expression* `enonce_deux_fois_deux_quatre` *en utilisant sa définition. Nous avons déjà vu plus haut que la commande* `simp` *demande à* LEAN *de « simplifier » l'expression. Dans tous les cas, si* LEAN *estime que l'égalité à montrer est évidente, la preuve se termine.*

4 Tactiques et écritures de preuves

Il existe diverses manières d'écrire une preuve en LEAN. Dans la méthode que nous allons étudier, l'écriture d'une preuve prend la forme de l'utilisation de **tactiques**, *qui imitent les éléments clés de l'écriture d'une preuve en langage naturel. Il existe beaucoup de tactiques différentes que nous allons apprendre au fur et à mesure de ces séances ; il sera à chaque fois très important de comprende le parallèle avec la façon dont on écrit une preuve en langage naturel.*

4.1 La tactique `intro`

Si l'objectif à démontrer est de la forme $\forall x \in E, P(x)$, écrit en LEAN de la façon suivante :

```
⊢ ∀ (x : E), P(x)
```

alors écrire `intro z` introduit dans le contexte un objet `z` de type `E`, et remplace l'objectif par `P(z)`. C'est similaire à la façon dont on montrerait la proposition $\forall x \in E, P(x)$: on écrirait dans la preuve : « Soit $x \in E$. Démontrons que $P(x)$ est vraie. ».

Le mot clé `intro` peut aussi servir si on doit montrer un objectif du type « $P \Rightarrow Q$ », écrit de la façon suivante :

```
⊢ P → Q
```

Alors écrire `intro hyp_P` introduit dans le contexte un objet `hyp_P`, correspondant à une hypothèse dont l'énoncé est `P`, et remplace l'objectif par `Q`. C'est similaire à la façon dont on écrirait une preuve de $P \Rightarrow Q$; on pourrait écrire : « Supposons que P est vraie, et notons `hyp_P` cette hypothèse. Démontrons que Q est vraie. ».

13. Tapez le code suivant, et comparez à la manière dont on écrirait la preuve si on devait la faire à la main. Pour chaque ligne de code, écrivez une phrase en langage naturel qui lui correspond.

```
theorem egalite_evidente : (∀ x : Int, (x = 3) → (x + 2 = 5)) := by
{
        intro x
        intro hypothese_x
        -- Preuve non-terminee
}
```

4.2 La tactique `rw`

La tactique `rw` *(« rewrite », « réécrire » en anglais), permet de réécrire une partie de l'objetif, notamment pour mener un calcul, ou expliciter une définition.*

Si le contexte contient une hypothèse de la forme suivante :

```
assertion : (expression_1) = (expression_2)
```

où `expression_1` *et* `expression_2` *représentent des expressions quelconques, alors écrire « * `rw [expression]` *», remplace dans l'objectif une occurence de l'expression* `expression_1` *par l'expression* `expression_2`*.*

Si on écrit plutôt `rw[←assertion]`, cela remplace une occurence de l'expression `expression_2` par `expression_1`.

14. A la suite des commentaires dans la réponse à la question précédente, tapez

```
rw [hypothese_x]
```

et regardez ce qui se produit. Comment conclure la preuve ?