Licence STS, semestre 4 — 2010–2011
Mathématiques pour l'Informatique (Info 229) — TP 1
`http://www.lri.fr/~paulin/MathInfo`

Mini-guide Coq

Interface

On utilise Coq via l'interface graphique CoqIDE (commande `coqide`). La fenêtre est organisée en trois zones :

À gauche on a le **script**. C'est là où on écrit, et c'est ce que l'on sauvera dans un fichier. On envoie progressivement le script à Coq (flèche bas) et on peut revenir en arrière (flèche haut) pour changer le script. Le script est composé de **déclarations** et de théorèmes, aussi appelés **buts**. Chaque théorème est suivi de sa **preuve**, composée d'une suite de **tactiques**. Chaque déclaration, but et tactique se **termine par un point**. On peut aussi y écrire des commentaires `(* comme ceci *)`.

En haut à droite, les **obligations de preuve** ou **buts** sont affichés quand on est en train de prouver un théorème. Il peut y avoir plusieurs buts. Le premier but est le seul à être détaillé. Par exemple :

```
x : nat
H : even x
============================
 even (x + 2)
```

Ici on a une variable `x`, une hypothèse `H` (qui énonce que `x` est pair) et il faut prouver `even (x+2)` (c'est à dire, `x+2` est pair).

Syntaxe

Correspondance entre notations papier et Coq

Papier	$\bot$	$\top$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$\forall x, P$	$\exists x, P$
Coq	`False`	`True`	`~P`	`P /\ Q`	`P \/ Q`	`P -> Q`	`forall x, P`	`exists x, P`

Une formule n'est bien formée que si les objets utilisés sont du bon type (les types jouent un rôle analogue à celui des ensembles). Par exemple, on ne peut pas écrire `(1 /\ 1)` ni `(P -> (P+2))`. Souvent, Coq devine et vérifie les types tout seul. Sinon, on peut les préciser avec la syntaxe `forall x : T,` ...

Quelques types utiles

`nat`	type des entiers naturels (`3 : nat`)
`Prop`	type des propositions logique (`False : Prop`)
`Set`	type des types de termes (`nat : Set`)
`T1 -> T2`	types des fonctions de `T1` dans `T2`
`T -> Prop`	types des prédicats unaires sur `T`

Déclarations

On utilise notamment les types dans les déclarations. La déclaration

– `Variable` ⟨`nom`⟩ `:` ⟨`type`⟩.

introduit un nouvel objet dans l'environnement nommé *nom* est dont le type est donné par l'expression *type*. Par exemple :

– `Variable p : nat -> Prop.` déclare un nouveau prédicat `p` portant sur un entier naturel

– `Variable T : Set.` déclare un nouveau type `T` de termes

– `Variable x : T.` déclare un habitant `x` dans le type `T`.

On peut aussi postuler une certaine propriété sur les objets introduits, on donne aussi un nom à cette propriété pour pouvoir l'utiliser ensuite dans les preuves.

– `Axiom` ⟨`nom`⟩ `:` ⟨`énoncé`⟩.

déclare l'énoncé comme un axiome, qui pourra ensuite être utilisé comme une hypothèse.

Finalement, pour éviter des notations trop lourdes, on peut introduire une *abréviation* et celle-ci peut (ou non) être paramétrée par des variables.

– `Definition` ⟨`nom`⟩ [`variables`] `:=` ⟨`corps`⟩.

introduit une définition pour la suite. Par exemple :

```
Definition f (x:nat) := x + 2.
Definition p_stable_f : Prop := forall x, p x -> p (f x).
Axiom ax1 : p_stable_f.
```

Tactiques

Coq est basé sur la déduction naturelle, mais les tactiques se comportent différemment pour plus de confort. Par exemple, quand on a en hypothèse `H` une conjonction `P /\ Q`, la tactique `destruct H` va ajouter directement `P` et `Q` comme nouvelles hypothèses et éliminer l'hypothèse `H` devenue inutile.

Exemple 1 (`intros`) *Supposons que la conclusion du but est* $\forall x \in \mathbb{N}, p(x) \Rightarrow \exists y, p(y)$*, c'est à dire qu'on a :*

```
...
============================
 forall x:nat, p x -> exists y:nat, p y
```

La tactique `intros` *va introduire la variable* `x` *ainsi que l'hypothèse* `p x`*. On aura un but de la forme :*

```
...
x : nat
H : p x
============================
 exists y:nat, p y
```

Exemple 2 (`exists`, `assumption`) *Sur le but obtenu dans l'exemple précédent, la tactique* `exists` *permet d'introduire la quantification existentielle, en fournissant un* témoin*. On fait ici* `exists x` *(on choisit d'instancier y par x) et on obtient le but suivant :*

```
...
x : nat
H : p x
============================
 p x
```

Concept	Utilisation
hypothèse	`assumption` permet de conclure quand le but est une hypothèse
$\forall$	introduction avec `intro` (ou `intros`), élimination avec `apply`
$\Rightarrow$	introduction avec `intro` (ou `intros`), élimination avec `apply`
$\neg$	introduction avec `intro` (ou `intros`), élimination avec `apply`
$\wedge$	intro : `split`, élim : `destruct H` (ou `destruct H as (H1,H2)`)
$\vee$	intro : `left` ou `right`, élim : `destruct H` (ou `destruct H as [H|H]`)
$\exists$	intro : `exists t`, élim : `destruct H` (ou `destruct H as (x,H)`)
$\top$	intro : `trivial`
$\bot$	élim : `trivial`
$t_1 = t_2$	introduction avec `reflexivity`, élimination avec `rewrite H`, `rewrite <- H` pour une réécriture dans le but et `rewrite H in H'`, `rewrite <- H in H'` pour une réécriture dans l'hypothèse `H'`
assertion	`assert P` permet d'introduire le résultat intermédiaire `P` qu'on devra prouver avant de l'avoir comme nouvelle hypothèse (choix du nom de l'hypothèse avec `assert P as H`)
définition	déroulage dans le but avec `unfold def`, dans une hypothèse avec `unfold def in H` (on peut donner plusieurs définitions d'un coup, séparées par des virgules)
calcul	forcer le calcul d'une opération (addition, concaténation, etc.) avec `simpl` qui s'utilise comme `unfold`
inductif	introduction avec `apply` et le nom de la règle, élimination avec `induction`
arithmétique	`omega` résout les buts "simples" reposant uniquement sur des (in)équations entre entiers
lemmes	un lemme s'utilise comme une hypothèse, par exemple on peut faire `apply lemme` ou `rewrite lemme`

On peut conclure par `assumption`.

Exemple 3 (`apply`) *Sur un but où la conclusion est Q et qu'on a une hypothèse (ou un lemme)* `H` *de la forme $P \Rightarrow Q$. on peut l'utiliser en faisant* `apply H`. *On se retrouvera avec à prouver le but P.*

La tactique ***`apply`*** *s'utilise de manière plus générale avec une hypothèse* ***`H`*** *de la forme. $\forall x_1 \ldots \forall x_k,\ P_1 \Rightarrow \ldots P_n \Rightarrow Q'$. Le système instancie les variables x_i pour que Q' corresponde au but à montrer et on se retrouvera avec n buts, un pour chaque P_i.*

```
H : forall x : nat, p (x - 1) -> p x
H0 : p 3
============================
p 4
```

La tactique ***`apply H.`*** *remplace x par 4 et retourne le nouveau but :*

```
H : forall x : nat, p (x - 1) -> p x
H0 : p 3
============================
p (4 - 1)
```

Il se peut qu'on doive spécifier les x_j avec la syntaxe `apply H with (x_j := t_j)...`

Exemple 4 (`elim`) *Si l'on a une hypothèse* `H` *qui postule $\neg P$ et qu'on sait prouver P, alors on peut conclure en éliminant la négation, indépendamment de la conclusion du but. Pour cela on fait* `elim H`, *et la conclusion du but devient P. En particulier, un but de la forme suivante est prouvé en faisant* `elim H` *suivi de* `assumption` *:*

```
...
H1 : ~ P
H2 : P
============================
 ...
```

Exemple 5 (`destruct`) *Considérons un but de cette forme :*

```
...
H : P
============================
 Q
```

La tactique ***`destruct H`*** *permet d'éliminer des conjonctions, disjonctions et existentiels selon l'hypothèse* `H` *visée. Si c'est une conjonction* `P1 /\ P2` *on se retrouve avec deux hypothèses au lieu d'une seule : une pour* `P1`, *l'autre pour* `P2`. *Si c'est une disjonction* `P1 \/ P2`, *on va obtenir deux sous-buts : dans le premier l'hypothèse* `H` *sera devenue* `P1` *; dans le second elle sera* `P2`.

Lemmes

Pour faire des raisonnements classiques non-constructifs, on chargera le module classique de Coq : `Require Import Classic`. On a ainsi accès à des lemmes utiles :

$$\begin{aligned} \texttt{NNPP} &: \forall P,\ \neg\neg P \Rightarrow P \\ \texttt{classic} &: \forall P,\ P \vee \neg P \\ \texttt{not\_all\_ex\_not} &: \forall P,\ (\neg\forall n, Pn) \Rightarrow \exists n, \neg Pn \end{aligned}$$

On peut vérifier l'énoncé d'un lemme grâce à la fenêtre `Check` : menu *Queries / Check*, on obtient une nouvelle zone en bas de la fenêtre.