Pense-bête

Symbole	Introduction	Elimination
$\forall$	`intro` ou `intros`	`apply` ⟨`Hyp`⟩
$\Rightarrow$	`intro` ou `intros`	`apply` ⟨`Hyp`⟩
$\neg$	`intro`	`destruct` ⟨`Hyp`⟩
$\wedge$	`split`	`destruct` ⟨`Hyp`⟩ `as (H1,H2)`
$\vee$	`left` ou `right`	`destruct` ⟨`Hyp`⟩ `as [H1|H2]`
$\exists$	`exists` ⟨`term`⟩	`destruct` ⟨`Hyp`⟩ `as (x,H)`
$\top$	`trivial`	$--$
$\bot$	$--$	`exfalso` ou `destruct` ⟨`Hyp`⟩
$t_1 = t_2$	`reflexivity`	`rewrite <-` ⟨`Hyp1`⟩ `in` ⟨`Hyp2`⟩

Les parties en gris sont optionnelles et permettent de nommer les hypothèses que l'on va introduire dans l'environnement. Dans le cas de la réécriture d'une égalité, la flèche vers la gauche permet d'inverser le sens de réécriture et le `in` permet de réécrire l'égalité dans une hypothèse.

Concept	Tactique	Utilisation
hypothèse	`assumption`	permet de conclure quand la conclusion du but courant est également une hypothèse de ce but.
étape	`assert` ⟨`Form`⟩ `as H`	permet d'introduire un résultat intermédiaire ⟨`Form`⟩ qu'on devra prouver avant de l'avoir comme nouvelle hypothèse.
définition	`unfold` ⟨`def`⟩ `in` ⟨`Hyp`⟩	déplie la définition ⟨`def`⟩.
calcul	`simpl` `in` ⟨`Hyp`⟩	permet d'effectuer un calcul (addition, concaténation...)
inductif	`apply` ⟨`règle`⟩	introduction correspondant à une règle de construction
	`induction` ⟨`Hyp`⟩	application du principe d'induction
	`inversion` ⟨`Hyp`⟩	raisonnement par cas
arithmétique	`omega`	résolution d'(in)équations entre entiers.

Détails des tactiques

— intros

La tactique `intros` applique la règle d'introduction correspondant au connecteur ou quantificateur de la conclusion du but courant. Et elle recommencera, autant de fois que possible, sur la conclusion du but obtenu. Par exemple, si la conclusion du but courant est : $\forall x, p(x) \Rightarrow \exists y, p(y)$, a tactique `intros` va introduire la variable `x` ainsi que l'hypothèse `p x`.

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

$\xrightarrow{\texttt{intros.}}$

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

— exists

Pour prouver une formule quantifiée existentiellement comme `exists y:nat, p y` , l'utilisateur doit fournir à Coq à la fois le *témoin* et la preuve que ce témoin vérifie le prédicat `p` (ce qui correspond à la règle d'introduction de $\exists$). La tactique `exists` permet de faire cela. Dans le but obtenu précédemment, on peut instancier `y` par `x` dans la formule que l'on cherche à prouver à l'aide de la commande `exists x`. Il restera alors à montrer `p x`.

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

— assumption

La tactique `assumption` correspond à la règle `axiome` de la déduction naturelle. On peut donc l'utiliser pour finir la preuve quand la conclusion du but courant se trouve dans les hypothèses.

```
x : nat
H : p x                     assumption.
============================   ⟶               Proof completed.
 p x
```

— apply

La tactique `apply` permet d'utiliser une formule que l'on a en hypothèse. Par exemple, si la conclusion du but courant est une formule `Q` et que l'on a en hypothèse une formule `P -> Q` (nommée `H`), alors on peut appliquer cette hypothèse grâce à la commande `apply H`. Il restera alors à prouver `P`.

```
H : P -> Q                  apply H.             H : P -> Q
============================   ⟶                 ============================
 Q                                                P
```

D'une manière plus générale, la tactique apply peut s'utiliser sur toute hypothèse `H` de la forme $\forall x_1 \ldots \forall x_k,\ P_1 \Rightarrow \ldots P_n \Rightarrow Q'$. Il est parfois nécessaire de préciser avec quelles valeurs il faut instancier les différentes variables `x_i`.
Ceci peut se faire à l'aide de la commande `apply H with (x_1 := y_1)` (...) `(x_n := y_n)`.

— destruct

La tactique `destruct` permet d'éliminer des conjonctions, disjonctions, négations, contradictions et existentiels en hypothèse. Si c'est une conjonction `H:P1 /\ P2` on se retrouve avec deux hypothèses au lieu d'une seule : une pour `P1` et l'autre pour `P2`.

```
                                                 H1 : P1
H : P1 /\ P2                destruct H.          H2 : P2
============================   ⟶                 ============================
 Q                                                Q
```

Si c'est une disjonction `P1 \/ P2`, on obtient deux sous-buts avec uniquement `P1` ou `P2` en hypothèse.

```
H : P1 \/ P2     destruct H.     H : P1                 H : P2
================     ⟶           ================   +   ================
 Q                                Q                      Q
```