Aide mémoire Coq pour les TP L1 semestre 1

Alexandre Tessier

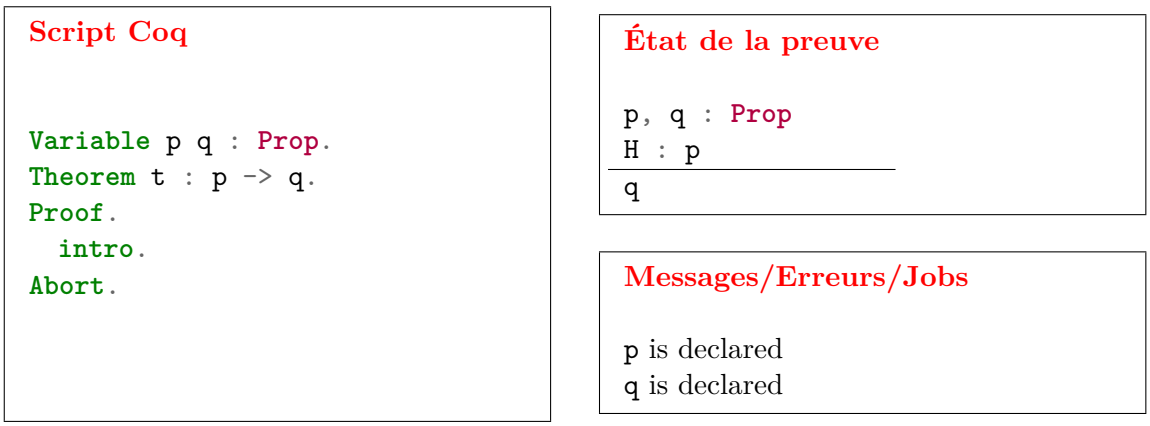
22 novembre 2023

1 Environnements Coq

Il existe différents environnements pour Coq. Ici nous utilisons l'un des deux environnements préconisés sur la page de l'assistant de preuve Coq de l'INRIA `https://coq.inria.fr/` :

1. L'IDE `coqide` de la plateforme **Coq Prover** à installer sur l'ordinateur :
   — installer `coq prover` depuis la logithèque ou simplement avec la commande
   ```
   sudo snap install coq-prover
   ```
   — pour les systèmes propriétaires, télécharger l'installeur sur la page `https://coq.inria.fr/download`
   Une fois **Coq Prover** installé, lancer l'IDE avec la commande `coqide` ou en ouvrant un fichier Coq depuis le gestionnaire de fichiers.
2. L'environnement `jsCoq` en ligne `https://coq.vercel.app/`

Dans ces deux environnements, la fenêtre est partagée en trois :
— à gauche, la fenêtre qui contient le script Coq
— en haut à droite, la fenêtre d'état de la preuve : les variables, les hypothèses disponibles, les buts à prouver...
— en bas à droite, la fenêtre des messages ou erreurs



2 Commentaires

Un commentaire coq commence par `(*` et se termine par `*)` :

```
(*
 * voici un commentaire
 * il sera ingoré par coq
 *)
```

3 Sections

On structure le script Coq en sections, chaque section est un environnement Coq indépendant.

```
Section Exemple.
  (* on place ici le contenu de la section *)
End Exemple.
```

4 Variables propositionnelles

Au début de la section, on déclare toutes les variables propositionnelles utilisées dans les formules.

```
  Variable p q : Prop.
```

Si vous utilisez `jsCoq` dans votre navigateur, `Prop` sera automatiquement remplacé par $\mathbb{P}$.

5 Connecteurs logiques, formules logiques

connecteur	*Coq*	*jsCoq*
$\top$	`True`	`True`
$\bot$	`False`	`False`
$\neg$	`~`	`~`
$\wedge$	`/\`	$\wedge$
$\vee$	`\/`	$\vee$
$\rightarrow$	`->`	$\rightarrow$

La dernière colonne indique la transformation faite automatiquement par `jsCoq`.
On utilisera également le parenthésage. Attention, toutes les parenthèses ne sont pas obligatoire en Coq (priorité $\neg$, $\wedge$, $\vee$, $\rightarrow$ et évaluation de droite à gauche). Exemple :

- `p /\ q \/ r` signifie `(p /\ q) \/ r`
- `p \/ q /\ r` signifie `p \/ (q /\ r)`
- `p -> q -> r` signifie `p -> (q -> r)`
- `p /\ q /\ r` signifie `p /\ (q /\ r)`

Si un identificateur termine pas un nombre, comme `exemple12`, `jsCoq` place automatique le nombre en indice $\texttt{exemple}_{12}$.

6 Preuve d'un théorème

On donne le théorème à prouver par `Theorem nomDuThéorème : laFormule.` suivi de la preuve qui commence par la commande `Proof.` et se terminera généralement par `Qed.`

```
Section Exemple.
  Variable p : Prop.
  Theorem exemple : p -> p.
  Proof.
    (* on place ici la preuve du théorème exemple *)
  Qed.
End Exemple.
```

On peut terminer une preuve en l'abandonnant avec la commande `Abort.` et passer à la suite.

```
Section Exemple.
  Variable p : Prop.
  Theorem exemple : p -> p.
  Proof.
    (* je n'arrive pas à faire la preuve du théorème exemple
     * mais je veux continuer sans avoir prouvé le théorème exemple
     *)
  Abort.
End Exemple.
```

7 Tactiques

Pour écrire les preuves, nous utiliserons des *tactiques* Coq. C'est un peu comme des règles de la déduction naturelle, pour introduire (dans le but) ou éliminer (dans l'hypothèse).
Voici les tactiques Coq que nous utiliserons dans nos scripts Coq pour prouver des théorèmes de la logique propositionnelle.

7.1 Axiome

But et Hypothèse. On doit prouver un but `A` et on dispose d'une hypothèse `A`. On utilise la tactique

```
assumption.
```

qui supprimera le but `A`.

7.2 Vrai

But. On doit prouver un but `True`. On utilise la tactique

```
trivial.
```

qui supprimera le but `True`.

7.3 Contradiction

Hypothèse. On doit prouver un but `A` et on a l'hypothèse `H : False`. On utilise la tactique

```
contradiction H.
```

qui supprimera le but `A`.

7.4 Implication

But. On doit prouver un but `A -> B`. On utilise la tactique

```
intro H.
```

qui ajoutera l'hypothèse `H : A` aux hypothèses et remplacera le but par `B`. Si on ne donne pas de nom à l'hyptohèse, en écrivant seulement

```
intro.
```

Coq la nommera automatiquement (`H`, `H0`, `H1`, etc).

```
Section Exemple2.
  Variable p : Prop.
  Theorem tauto1 : p -> p.
  Proof.
    intro Hypo1.
    assumption.
  Qed.
End Exemple2.
```

Hypothèse. On doit prouver un but `B` et on a l'hypothèse `H : A -> B`. On utilise la tactique

```
apply H.
```

qui remplacera le but `B` par le but `A`.

7.5 Conjonction

But. On doit prouver un but `A /\ B`. On utilise la tactique

```
split.
```

qui remplacera le but `A /\ B` par deux nouveaux buts `A` et `B`.

Hypothèse. On a l'hypothèse `H : A /\ B`. On utilise la tactique

```
destruct H as (HGauche,HDroite).
```

qui remplacera `H : A /\ B` par `HGauche : A` et `HDroite : B`. Si on ne précise pas les noms des deux nouvelles hypothèses, Coq les nomme automatiquement.

7.6 Disjonction

But. On doit prouver un but `A \/ B`. Il nous suffit de prouver `A` ou `B`. On indique laquelle des deux formules on souhaite prouver avec la tactique

```
left.
```

qui remplacera le but `A \/ B` par le but `A`, ou la tactique

```
right.
```

qui remplacera le but `A \/ B` par le but `B`.

Hypothèse. On doit prouver un but `C` et on a l'hypothèse `H : A \/ B`. On utilise la tactique

```
case H.
```

qui remplacera le but `C` par deux nouveaux buts `A -> C` et `B -> C`.

```
Section Exemple3.
  Variable p q r : Prop.
  Theorem t1 : ((p /\ q) \/ (q /\ r)) -> q.
  Proof.
    intro H1.
    case H1.
    intro H2.
    destruct H2 as (H3,H4).
    assumption.
    intro H5.
    destruct H5 as (H6,H7).
    assumption.
  Qed.
End Exemple3.
```

On peut faire apparaître explicitement la preuve des deux sous-buts en utilisant les puces -, +, * :

```
Section Exemple4.
  Variable p q r : Prop.
  Theorem t2 : ((p /\ q) \/ (q /\ r)) -> q.
  Proof.
    intro H1.
    case H1.
    - intro H2.
      destruct H2 as (H3,H4).
      assumption.
    - intro H5.
      destruct H5 as (H6,H7).
      assumption.
  Qed.
End Exemple4.
```

7.7 Négation

Coq peut traiter la formule `~A` comme s'il s'agissait de `A -> False`. De même, il peut traiter la formule `A` comme s'il s'agissait de `True -> A`.
Pour la négation `~A` nous utiliserons donc les mêmes tactiques que pour l'implication `A -> False`.

But. On doit prouver un but `~A`. Il nous suffit d'utiliser la tactique

```
intro H.
```

qui remplacera le but `~A` par le but `False` et ajoutera l'hyptohèse `H : A`.

Hypothèse.

1. On doit prouver le but `B` et on a l'hypothèse `H : ~A`. On utilise la tactique

   ```
   destruct H.
   ```

   qui remplace le but `B` par le but `A` et supprime l'hypothèse `H : ~A`.
2. On doit prouver le but `False` et on a l'hypothèse `H : ~A`. On utilise la tactique

   ```
   apply H.
   ```

   qui remplace le but `False` par le but `A`.
3. On doit prouver le but `B` et on a les hypothèses `H1 : A` et `H2 : ~A`. On utilise la tactique

   ```
   contradiction.
   ```

   qui supprime le but `B`.

7.8 Récupitulatif

connecteur	Coq	introdution (dans le but)	élimination (dans l'hypothèse `H`)
$\top$	`True`	`trivial.`	
$\bot$	`False`		`contradiction H.`
$\neg$	`~`	`intro.`	`apply H.` ou `destruct H.` ou `contradiction H.`
$\wedge$	`/\`	`split.`	`destruct H as (H1,H2).`
$\vee$	`\/`	`left.` ou `right.`	`case H.`
$\rightarrow$	`->`	`intro.`	`apply H.`