UFR IM²AG
UNIVERSITÉ Grenoble Alpes

Modèles de Calcul [Lambda-Calcul]
Preuves Coq pour le λ-calcul simplement typé

Martin Bodin, Pascal Fradet, Jean-François Monin

Cette fiche comporte des exercices permettant de découvrir quelques possibilités de Coq comme assistant à la preuve.

Reprendre le fichier Coq utilisé pour le typage simple.

1 Réductions pas à pas en mode preuve interactive

Pour détailler pas à pas le processus de réduction déclenché par un calcul, on va utiliser le mode preuve interactive de Coq qui fonctionne ainsi : on énonce un `Theorem` (un théorème à démontrer) puis on enchaine l'application de *tactiques* élémentaires qui correspondent à l'usage de règles valides jusqu'à ce qu'il n'y ait plus rien à démontrer.

Illustrons l'idée sur le calcul de `cnot ctr` : on va démontrer l'égalité `cnot ctr = cfa` en utilisant des tactiques de réduction élémentaires – elles porteront sur premier membre puisque dans le second il n'y a rien à réduire. Notre objectif est d'atteindre le but `cfa = cfa` qui pourra être démontré par la tactique `reflexivity`. Auparavant, notre première tactique s'appelle `cbv` pour *Call By Value*. Nous allons l'utiliser avec, en argument, un nom de réduction qui sera `beta` ou `delta`. Suivant le cas, des β-réductions ou des δ-réductions (expansion de définitions) sont effectuées sur les termes du théorème à démontrer. Pour `delta`, on peut en option (avec `[...]`) spécifier le ou les noms de constantes à expanser (par défaut toutes les définitions sont expansées). Sur notre exemple, on obtient la preuve suivante :

```
Theorem not_true : cnot ctr = cfa.
Proof. (* marque le debut de la preuve *)
  cbv delta [cnot]. (* les tactiques commencent par des minuscules *)
  cbv beta.
  cbv delta [ctr].
  cbv beta.
  cbv delta [cfa].
  (* A ce stade on a une egalite entre deux termes syntaxiquement identiques *)
  reflexivity.
Qed. (* marque la fin de la preuve *)
```

1. Exécuter cette preuve, pas à pas, et observer les effets de chaque tactique.
2. Qu'est-ce que la δ-réduction ?
3. Qu'est-ce que la β-réduction ?
4. Produire une preuve analogue pour démontrer `cnot cfa = ctr`.
5. Observer qu'en fait, invoquer `reflexivity` dès le début suffirait pour faire cette démonstration : cet emploi de `reflexivity` cache donc des δ-réductions et des β-réductions.
6. Démontrer une ou plusieurs des propriétés suivantes (table de vérité du `and`) :
   `cand cfa cfa = cfa`, `cand cfa ctr = cfa`, `cand ctr cfa = cfa`, `cand ctr ctr = ctr`.
7. Démontrer les propriétés de la table de vérité du `or`.

8. Démontrer
   `Remark cS_compo : forall n, cS n = fun f => f  (n f).`
   puis
   `Theorem cS_is_successor : forall n, cS [n]N = [S n]N.`

2 Lois algébriques sur les booléens

Les opérations booléennes satisfont de nombreuses lois algébriques (commutativité, associativité, distributivité, etc.) et leur démonstration permet d'en découvrir davantage sur l'assistant à la preuve Coq.

1. On va d'abord s'intéresser à la commutativité de la conjonction : $a \wedge b = b \wedge a$. Implicitement, on a ici une quantification universelle sur a et sur b. En Coq on l'exprimera ainsi :

   ```
   Theorem cand_comm : forall a b : cbool, cand a b = cand b a.
   Proof.
   ```

   Rappel : pour démontrer $\forall x, \mathrm{P}(x)$, on démontre $\mathrm{P}(x)$ sur un x_0 arbitraire. La règle de déduction naturelle utilisée est une introduction de $\forall$, et en Coq on utilise une tactique appelée simplement `intro` prenant en argument le nom de la variable libre considérée, par exemple a, a_0, etc. On a ici deux quantifications universelles, on peut donc écrire `intro a. intro b.` ou, en abrégeant : `intros a b`. On peut ensuite procéder aux réductions.

   ```
   Theorem cand_comm : forall a b : cbool, cand a b = cand b a.
   Proof.
     intros a b. cbv.
   ```

   Tenter d'exécuter cette preuve et de continuer. Qu'observe-t-on ?

2. Ici, `reflexivity` ne va donc pas fonctionner.
   Pour prendre un exemple plus simple, sur des fonctions unaires f et g, il est en général faux que $\lambda x.g\,(f\,x)$ soit égal à $\lambda x.f\,(g\,x)$. Mais cela peut arriver sur des fonctions particulières, par exemple si f est la fonction identité.
   Pour revenir à notre cas, on espère que l'égalité est prouvable lorsque a et b sont parmi les fonctions $\lambda xy.x$ et $\lambda xy.y$.

3. Une première façon de procéder consiste à énoncer et démontrer ces théorèmes dans chaque combinaison particulière, par exemple avec respectivement `ctr` et `cfa` pour a et b.
   Traduire en Coq l'énoncé `true` $\wedge$ `false` = `false` $\wedge$ `true` puis le démontrer.

Avertissement. *Les questions suivantes sont étoilées, ce qui signifie : de difficulté croissante (suivant le nombre d'étoiles) ; et optionnelles, au sens où elles ne sont pas indispensables au démarrage ou à la compréhension des fiches d'exercices suivantes. Il est recommandé aux étudiants en retard de démarrer les fiches suivantes et de revenir à cette fiche au besoin, ou pour les révisions.*

4. **(*)** Pour avoir une formulation générale des 4 énoncés particuliers en un seul théorème, une première possibilité serait

   ```
   Theorem cand_comm : forall a b : cbool,
     (a = ctr \/ a = cfa) -> (b = ctr \/ b = cfa) -> cand a b = cand b a.
   ```

   On va exprimer la même chose autrement, au moyen d'un procédé à la fois plus puissant et plus simple à manier. Pour cela on définit une numérotation des booléens, au moyen d'un type énuméré à 2 valeurs sur lequel on pourra ensuite raisonner par cas.

   ```
   (* Numerotation des booleens, pour enoncer les lois algebriques
      dans le cas general *)
   (* Definition d'un type enumerant les numeros *)
   Inductive bool_num : Set :=  bnt | bnf.
   ```

```
(* Fonction d'association entre numero et booleen *)
Definition cbool_of := fun n =>
  match n with
  | bnt => ctr
  | bnf => cfa
  end.
```

Observer les deux évaluations suivantes (`unfold X` est un raccourci pour `cbv delta [X]`) :
— `Eval unfold cbool_of in cbool_of bnt.`
— `Compute cbool_of bnt.`
Nous allons utiliser ces définitions pour démontrer différentes lois algébriques.

5. **(*)** La démonstration de la commutativité de la conjonction sur tous les termes ainsi énumérés est la suivante (avec l'énoncé du théorème) :

```
Theorem cand_comm :
  forall na nb,
  cand (cbool_of na) (cbool_of nb) = cand (cbool_of nb) (cbool_of na).
Proof.
intros na nb.
(* Preuve par cas sur na, en utilisant destruct *)
destruct na.
  (* On enchaine avec une preuve par cas sur nb *)
  - destruct nb. (* structuration de la preuve en niveaux,
                    niveaux max, avec les -, + et * *)
     (* 1er cas : unfolding de la definition de cbool_of, pour montrer *)
    + unfold cbool_of. reflexivity.
     (* 2e cas : en fait, reflexivity suffit *)
    + reflexivity.
    (* on peut aussi demander la meme chose dans tous les cas, en utilisant ";".
       Dans toutes les branches du destruct nb, on applique reflexivity *)
  - destruct nb; reflexivity.
Qed.
```

Exécuter cette preuve.

6. **(**)** Sur ce modèle, démontrer les lois de Morgan $\neg a \wedge \neg b = \neg(a \vee b)$ et $\neg a \vee \neg b = \neg(a \wedge b)$.
7. **(**)** Peut-on démontrer les lois de Morgan en utilisant l'approche directe ?
8. **(***)** Énoncer et démontrer à volonté d'autres lois algébriques sur $\wedge, \vee, \neg$:
idempotence, associativité, éléments neutres, absorbants, distributivité,...

3 Entiers de Church avec typage simple

1. **(*)** Énoncer et démontrer l'associativité de l'addition.
2. Le test à zéro d'un entier de Church est défini par : $\lambda n.\lambda x\, y.n\ (\lambda z.y)\ x$.
   Coder en Coq cette fonction `tz` en utilisant le type `cbool` des booléens de Church comme type résultat de cette fonction de test à zéro.
3. **(*)** Prouver que `tz` renvoie `ctr` pour l'entier de Church `[0]N` et `cfa` pour n'importe quel autre entier de Church.