Modèles de Calcul [Lambda-Calcul]
Fiche 2 : Programmation en λ-calcul (les booléens)

Pascal Fradet, Jean-François Monin, Catherine Parent-Vigouroux

Rappels

Les booléens sont définis par `true` $\stackrel{\text{def}}{=} \lambda xy.x$ et `false` $\stackrel{\text{def}}{=} \lambda xy.y$.
Le `if` est codé par `cif` $\stackrel{\text{def}}{=} \lambda bmn.(b\ m\ n)$ (intuition : `if b then m else n` est encodé par $b\ m\ n$).

1 Codage des booléens et opérateur if

Nous allons étudier et jouer avec ce codage des booléens en Coq. On les appellera les booléens de Church.

```
Section type_booleen.

Variable T: Set.

(* les booleens sont des fonctions a deux parametres *)
Definition cbool := T -> T -> T.

(* codage de true *)
Definition ctr : cbool := fun x y => x.

(* codage de false *)
Definition cfa : cbool := fun x y => y.
```

1. Coder en Coq le `if`.
2. Vérifier (en utilisant `Compute`) que l'évaluation de `cif` sur *true* (`ctr`) et sur *false* (`cfa`) est correcte.

2 Codage des opérateurs booléens

On veut coder les trois opérateurs booléens : `not`, `and` et `or`.
Les lambda-termes correspondants sont :
- `cnot` $\stackrel{\text{def}}{=} \lambda bxy.b\ y\ x$
- `cand` $\stackrel{\text{def}}{=} \lambda abxy.a\ (b\ x\ y)\ y$
- `cor` $\stackrel{\text{def}}{=} \lambda abxy.a\ x\ (b\ x\ y)$

1. Coder ces trois opérateurs en Coq.
2. Vérifier avec `Compute` que `cnot ctr` se réduit bien en `cfa`.
3. On va maintenant détailler pas à pas le processus de réduction déclenché par ce calcul. On cherche à démontrer avec Coq que `cnot ctr = cfa`. On définit un `Theorem` (un théorème à démontrer) et on enchaine l'application de *tactiques* élémentaires. En l'occurrence, notre première tactique effectue des réductions : son nom est `cbv` pour Call By Value. Nous allons 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 `[` et `]`) 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 *)
```

   Exécuter cette preuve, pas à pas, et observer les effets de chaque tactique. Comprendre la tactique `reflexivity`.
4. Qu'est-ce que la δ-réduction ?
5. Qu'est-ce que la β-réduction ?
6. Reproduire la même preuve pour démontrer `cnot cfa = ctr`.
7. Observer qu'en fait, `reflexivity` suffirait pour faire cette démonstration et que, donc, cette tactique cache des δ-réductions et des β-réductions.
8. Coder l'opérateur `cand`.
9. Démontrer les propriétés suivantes (table de vérité du `and`) : `cand cfa cfa = cfa`, `cand cfa ctr = cfa`, `cand ctr cfa = cfa`, `cand ctr ctr = ctr`.
10. Coder l'opérateur `cor`.
11. Démontrer les propriétés de la table de vérité du `or`.

3 Lois algébriques

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 $\forall x, P(x)$, on démontre $P(x)$ 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 `true` et `false` pour a et b.

   Énoncer et démontrer en Coq : $\texttt{true} \wedge \texttt{false} = \texttt{false} \wedge \texttt{true}$.

4. Pour avoir une formulation générale des 4 énoncés particuliers en un seul théorème, on va définir 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 pouvoir definir les lois de Morgan
     dans le cadre general *)
  (* Definition d'un type enumerant les numeros *)
Inductive bool_num : Set :=  bnt | bnf.

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

   Observer les deux évaluations suivantes :
   — `Eval unfold cbool_of in cbool_of bnt.`
   — `Compute cbool_of bnt.`

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,...