Vincent Simonet (http://cristal.inria.fr/~simonet/)

Un peu de logique

Pour cette avant-dernière séance, nous allons manipuler des expressions logiques en Caml. Dans la première partie, nous allons écrire un algorithme de recherche exhaustive *permettant de tester si une expression est satisfiable (et de même s'il s'agit d'une tautologie). Dans la seconde partie, nous l'utiliserons pour résoudre quelques petits problèmes de logique simples.*

1 Expressions logiques

Les expressions logiques que nous manipulons sont formées à partir de :
– Deux constantes 0 et 1.
– Des variables propositionelles : $x_0, x_1, \ldots$.
– Un opérateur unaire : *non*
– Cinq opérateurs binaires : *ou*, *et*, *oux* (ou exclusif), *implique* et *equivaut*.

Nous ne donnons pas de définition formelle de telles expressions (une telle définition ayant été vue en cours). Cette syntaxe permet de construire des expressions telles que :

$$(x_1 \textit{ ou } x_2) \textit{ et } (x_3 \textit{ implique } 1)$$

$$(x_1 \textit{ ou } x_2) \textit{ et } (x_2 \textit{ ou } x_3) \textit{ et } x_4$$

Nous allons représenter les expressions logiques en *Caml* par des arbres syntaxiques. Pour cela, nous définissons un type somme récursif expr :

```
type expr =
    Vrai
  | Faux
  | X of int
  | Non of expr
  | Ou of expr * expr
  | Et of expr * expr
  | Oux of expr * expr
  | Implique of expr * expr
  | Equivaut of expr * expr
;;
```

▶ **Question 1** *Écrivez les expressions données ci-dessus en exemple sous forme d'arbres de type* expr.

Un environnement est une fonction de l'ensemble des variables $\{x_0, x_1, \ldots\}$ dans l'espace booléen. Dans la pratique, le support des environnements sera fini. Nous pourrons donc représenter un environnement en *Caml* par un tableau de booléens g. (Le contenu de la case $g.(i)$ correspondant à la valeur booléeene attribuée à la variable x_i.)

▶ **Question 2** *Définissez une fonction* eval *qui prend pour arguments une expression e (de type* expr*) et un environnement g. Cette fonction retournera un booléen correspondant à l'évaluation de e dans l'environnement g.*

value eval : $\rightarrow$*expr*$\rightarrow$*bool vectbool*

Nous cherchons maintenant à écrire une fonction qui, étant donnée une expression e, trouve un environnement g qui satisfait e. Pour cela, nous commençons par écrire deux fonctions liminaires.

▶ **Question 3** *Écrivez une fonction* incr *qui prend pour argument un tableau g de booléens (représentant un environnement). Votre fonction modifiera g en place de manière à obtenir l'enregistrement suivant pour l'ordre lexicographique. Si le tableau passé en argument est maximal pour cet ordre, vous lèverez une exception.*

value incr : $\rightarrow$*bool vectunit*

▶ **Question 4** *Écrivez une fonction* max_x *qui étant donnée une expression e calcule l'indice maximal de variable propositionelle qu'elle comporte.*

value max_x : $\rightarrow$*exprint*

▶ **Question 5** *Déduisez-en une fonction* satisfy *qui étant donnée un expression e retourne un environne-*

ment g qui satisfait e. Si l'expression e n'est pas satisfiable, vous lèverez une exception.

value satisfy : $\rightarrow expr\, bool\ vect$

Tautologie et satisfiabilité sont deux notions intimement liées : une expression e est une tautologie si et seulement si sa négation n'est pas satisfiable. Nous allons écrire une fonction tautologic qui donne son résultat sous une forme différente de la fonction satisfy (i.e. sans lever d'exception). En *Caml*, on dispose d'un type « option » prédéfini de la sorte :

```
type 'a option =
    None
  | Some of 'a
;;
```

Étant donnée une expression e, notre fonction tautologie retournera None si e est effectivement une tautologie (aucun contre-exemple trouvé), sinon Some g (où g est un environnement ne satisfaisant pas e).

▶ **Question 6** *Définissez en Caml la fonction* tautologic.

value tautologic : $\rightarrow expr\, bool\ vect\ option$

▶ **Question 7** *Quelle est la complexité de vos fonctions* satisfy *et* tautologie *en fonction du nombre de la taille des expressions et/ou du nombre de variables qu'elles font apparaître ?*

2 Petits problèmes

2.1 L'Île aux questions

Quelque part au delà des mers se dresse une île étrange appelée l'*Île aux Questions*. Son nom lui vient de la façon originale dont s'expriment les indigènes qui, pour toutes paroles, se contentent de poser des questions auxquelles il faut répondre par oui ou par non. Les habitants se répartissent en deux types : les *Positifs* et les *Négatifs*. Les premiers ne peuvent poser que des questions dont la réponse exacte est oui. Quand aux seconds, c'est le contraire, la réponse à leurs questions doit toujours être non.

▶ **Question 8** *En arrivant sur l'île, vous rencontrez deux habitants. L'un d'entre-eux demande « L'un de nous est-il négatif ? ». Quelle est la nature des deux personnages ?*

Ce problème est très simple à résoudre de tête mais nous allons essayer d'utiliser le résultat de la section 1. Commençons par nommer A le personnage qui parle et B l'autre. On modélise alors le problème de la façon suivante : la variable x_0 correspond à l'affirmation « *A est un positif* » et x_1 à « *B est un positif* ». Il suffit alors de déterminer si la proposition x_0 *equivaut* $((non\, x_0)$ *ou* $(non\, x_1))$ est satisfiable.

▶ **Question 9** *Que pensez-vous d'un indigène qui vous interroge : « Suis-je un positif ? ». Et d'un autre qui vous demande s'il est négatif ?*
Un habitant de l'île nommé Zorn m'a posé la question suivante : « Suis-je de ceux qui peuvent demander s'ils sont négatifs ? ». Peut-on en déduire quelque chose concernant Zorn ou bien cette histoire est-elle tout simplement impossible ?

▶ **Question 10** *Poursuivant votre chemin, vous rencontrez à nouveau deux personnages, A et B. Le premier demande au second « Suis-je négatif et toi positif ? ». Pouvez-vous en déduire ce que sont A et B ?*

▶ **Question 11** *Vous voilà maintenant face à trois indigènes A, B et C. A vous demande « B est-il négatif ? » alors que B se demande si A et C sont de la même espèce. Peut-on dire ce que sont A, B et C ?*

▶ **Question 12** *Une femme interroge son mari : « Sommes-nous de types différents ? ». Que pouvez-vous en déduire ?*

2.2 Enquête

Un commerçant de Londres appela Scotland Yard pour signaler que sa boutique venait d'être cambriolée. Trois suspects A, B et C furent interpellés et on prouva que :

1. Seuls A, B et C étaient entrés dans la boutique.
2. Si A était coupable, il n'avait pas plus d'un complice.
3. Si B était innocent, C l'était aussi.
4. Si il y avait exactement deux coupables, A était parmis ceux-là.
5. Si C était innocent, B l'était aussi.

▶ **Question 13** *Qui a-t-on inculpé ?*

2.3 Trois coffrets

La belle Portia a trois coffrets : l'un en or, l'autre en argent et le troisième en plomb. Ele a caché un de ses protraits dans l'un des coffrets. On peut déchiffrer sur ceux-ci les inscriptions suivantes :

Or	**Plomb**	**Argent**
Le portrait est ici.	Au moins deux des coffrets ont été gravés par Cellini.	Le portrait est ici.

▶ **Question 14** *Sachant que chaque coffret a été gravé par Bellini ou par Cellini et en vous rappelant que Bellini ne gravait que des vérités alors que Cellini ne ciselait que des mensonges, pouvez-vous dire où est le portrait et qui a gravé chacun des coffrets ?*

▶ **MPSI – Option Informatique**
Année 2002, Neuvième TP Caml
Vincent Simonet (http://cristal.inria.fr/~simonet/)

Un peu de logique

Un corrigé

▶ Question 2

```
let rec eval g = function
    Vrai -> true
  | Faux -> false
  | X i -> g.(i)
  | Non e -> not (eval g e)
  | Ou (e1, e2) -> (eval g e1) or (eval g e2)
  | Et (e1, e2) -> (eval g e1) && (eval g e2)
  | Oux (e1, e2) -> (eval g e1) <> (eval g e2)
  | Implique (e1, e2) -> not (eval g e1) || (eval g e2)
  | Equivaut (e1, e2) -> (eval g e1) = (eval g e2)
;;
```

▶ Question 3

```
exception Fin;;

let incr g =
  let n = vect_length g in
  let i = ref 0 in

  while !i < n && g.(!i) do
    g.(!i) <- false;
    i := !i + 1
  done;

  if !i = n then raise Fin
  else g.(!i) <- true
;;
```

▶ Question 4

```
let rec max_x = function
    Vrai | Faux -> 0
  | X i -> i
  | Non e -> max_x e
  | Ou (e1, e2) -> max (max_x e1) (max_x e2)
  | Et (e1, e2) -> max (max_x e1) (max_x e2)
  | Oux (e1, e2) -> max (max_x e1) (max_x e2)
  | Implique (e1, e2) -> max (max_x e1) (max_x e2)
  | Equivaut (e1, e2) -> max (max_x e1) (max_x e2)
;;
```

▶ Question 5

```
exception Insatisfiable;;

let satisfy e =
  let n = max_x e in
  let g = make_vect (n+1) false in

  try
    while not (eval g e) do
      incr g
    done;
    g
  with
    Fin -> raise Insatisfiable
;;
```

▶ Question 6

```
let tautologic e =
  let n = max_x e in
  let g = make_vect (n+1) false in

  try
    while eval g e do
      incr g
    done;
    Some g
  with
    Fin -> None
;;
```