

**Rapport de projet de Programmation Fonctionnelle**

Ryan Yala & Mohamed Aziz Taieb

17 mars 2022

Solveur Booléen en OCAML

**Encadrant :** Christian Codognet

Année 2022-2023

**1 -Rappel de sujet :**

* Implémentation en OCAML d’un solveur booléen, c’est-à-dire un Algorithme permettant de résoudre l’ensemble des solutions d’un système d’équations booléennes.
* La simplification d’équations booléennes peut utiliser différentes méthodes : outre les classiques développement via associativité, commutativité, distributivité, etc. Les tables de vérité ou les diagrammes de Venn permettent une bonne vue de l’ensembles des expressions. Dans le cadre de ce projet, c’est l’implémentation des tables de vérité qui ont permis la résolution de système d’équations Booléennes.
* Une fois la table de vérité nous avons choisit de substitué chaque variable afin d’évaluer la satisfaction d’une équation donnée dans un environnement donné. Enfin nous avons généralisé pour tout le système.

**2 -Présentation des structures de données :**

* **Chaque équation est représentée par le type suivant :**

**type eq = Equals of eb \* eb ;;**

C’est-à-dire un couple d’expression booléenne Equals(a , b) avec a le membre de gauche et b le membre de droite de l’égalité. Autrement dit Equals(a , b) ⬄ a=b avec a et b des expressions booléennes de type eb .

* **Chaque Système d’équations est représenté par le type suivant :**

**[Equals(a,b) ;Equals(c,d) ; . . . ]**

C’est-à-dire une liste d’égalité de type eq .

**3 -Présentation des Algorithmes mis en œuvre :**

* **Détermination des variables du système d’équation :** Nous avons effectué une recherche simple sur une liste (système d’équations) qui récupère toutes les variables et les mets dans une autre liste puis enfin nous avons supprimé toutes les occurrences multiples de cette dernière**.**
* **Génération de tous les environnements possibles :** Nous avons implémenté un algorithme de type diviser pour Reigner qui prend en entrée une liste de variable, la découpe en deux sous-listes et ajoute à la première pour chaque combinaison possible la valeur vraie, puis ajoute à la seconde pour chaque combinaison possible la valeur Faux.
* **Evaluation de la satisfaction d’une équation :**  Nous avons choisi de substitué chaque variable afin d’évaluer la satisfaction d’une équation donnée dans un environnement donné. Enfin nous avons généralisé pour tout le système.

**4 -Problèmes rencontrés et Solutions :**

Le premier problème rencontré était le choix de structure de données pour représenter l’ensemble d’équation. On voulait absolument présenter le système comme un ensemble commençant par ‘{‘ et finissant par ‘}’ , alors on a passé du temps à chercher une façon de définir ce type de manière que sa définition nous aide à effectuer la récursion . Après une heure de recherche et d’essai, nous nous sommes rendu compte que le type était inutile dans le contexte de la programmation fonctionnelle, nous nous sommes alors suffit par une liste d’équation définit par Equals afin de faciliter la lecture du code.

Le deuxième problème envisagé était la réalisation de la fonction qui construit l’environnement. Nous avions pris du temps à trouver une manière de l’écrire. D’abord nous avons voulu faire la fonction avec des boucles ‘for’ et calculer le nombre de variables mais cela nous éloignait du cadre de la programmation fonctionnelle et ne correspond pas à ce qu’on a appris en TD donc on avait effectué quelques tables de vérité sur papier et nous nous sommes rendu compte qu’une récursion peut effectivement être faite. Avec un ‘let in’ bien placé et des conditions de base bien choisit, on a créé une fonction simple et efficace.

Enfin le troisième problème rencontré était la planification du troisième exercice, on avait plusieurs manières d’évaluer l’ensemble donc on avait du mal à choisir une technique. Nous nous sommes contentés de créer plusieurs fonctions qui traitent des sous problèmes et les appeler dans des autres fonctions pour traiter des problèmes plus grands, ce qui a permis d’effectuer un code clair et net .

**5 -Jeux d’essai**

La fonction finale du projet est appelée solveur. Voici quelques essais sur cette fonction et la fonction génératrice de la table de vérité :

1) {V(1) XOR V(2) = VRAI; V(2) OR V(3) = V(1); V(3) AND V(1) =FAUX }

Construction d’une liste de variable :

En OCAML:

listevar [Equals(XOR(V(1),V(3)),VRAI);Equals(OR(V(2),V(3)),V(1));Equals(AND(V(3),V(1)),FAUX)];;

**Résultat :**

[V(1);V(2);V(3)]

2) {V(1) XOR V(2) = VRAI; V(2) OR V(3) = V(1); V(3) AND V(1) =FAUX }

Construction de l’environnement :

En OCAML:

listeenv [V(1);V(2);V(3)] ;;

**Résultat :**

[[(V 1, VRAI); (V 2, VRAI); (V 3, VRAI)];

[(V 1, VRAI); (V 2, VRAI); (V 3, FAUX)];

[(V 1, VRAI); (V 2, FAUX); (V 3, VRAI)];

[(V 1, VRAI); (V 2, FAUX); (V 3, FAUX)];

[(V 1, FAUX); (V 2, VRAI); (V 3, VRAI)];

[(V 1, FAUX); (V 2, VRAI); (V 3, FAUX)];

[(V 1, FAUX); (V 2, FAUX); (V 3, VRAI)];

[(V 1, FAUX); (V 2, FAUX); (V 3, FAUX)]]

3) {V(1) XOR V(2) = VRAI; V(2) OR V(3) = V(1); V(3) AND V(1) =FAUX }

Résoudre l’équation :

En OCAML :

solveur [Equals(XOR(V(1),V(3)),VRAI);Equals(OR(V(2),V(3)),V(1));Equals(AND(V(3),V(1)),FAUX)];;

**Résultat :**

[[(V 2, VRAI); (V 3, FAUX); (V 1, VRAI)]]

4) On essaie une antilogie

{V(1)=NOT(V(1)}

En OCAML:

solveur [Equals(V(1),NOT(V(1)))] ;;

**Résultat :**

[]

Code OCAML

type eb= V of int | VRAI | FAUX | AND of eb \*eb | OR of eb \* eb | XOR of eb \* eb | NAND of eb \* eb | NOT of eb ;;

(\*type défini pour faciliter la lecture du système d’équation\*)

type eq = Equals of eb \* eb ;;

(\*append fonction triviale\*)

let rec concat l1 l2 = match l1 with

[]->l2

|x1::ll1->x1::(concat ll1 l2) ;;

(\*appartient fonction qui renvoie true en cas d'appartenance false sinon \*)

let rec appartient x l1 = match l1 with

[]->false

|t1::q1-> if t1 = x

then true

else

appartient x q1;;

(\*Fonction qui renvoie toutes les occurences de toutes les variables présentes dans une expression booléenne\*)

let rec listevareb e = match e with

V(a) -> [V(a)]

|VRAI -> []

|FAUX -> []

|AND(a,b) -> concat (listevareb a) (listevareb b)

|OR(a,b) -> concat (listevareb a) (listevareb b)

|XOR(a,b) -> concat (listevareb a) (listevareb b)

|NAND(a,b) -> concat (listevareb a) (listevareb b)

|NOT(a) -> listevareb a ;;

(\*Fonction qui renvoie toutes les occurrences de toutes les variables présentes dans une Équation booléenne\*)

let listevareq e =match e with

Equals(a,b) -> concat (listevareb a) (listevareb b) ;;

(\*Fonction qui renvoie toutes les occurrences de toutes les variables présentes dans une liste d'équations booléenne\*)

let rec listevarens e = match e with

[]-> []

|a::l -> concat (listevareq a) (listevarens l) ;;

(\*Fonction qui prend en argument une liste l1 de variable et une liste accumulateur l2 et qui met dans l2 au plus une occurrence de chaque variables de l1\*)

let rec listevaruni l1 l2 = match l1 with

[]-> l2

|a::ll1 -> if appartient a l2 =true then listevaruni ll1 l2

else listevaruni ll1 (a::l2);;

(\*Fonction final exercice 1 qui prend en argument une Liste d’équation et renvoie une liste de toutes les inconnue du système \*)

let listevar e = listevaruni (listevarens e) [];;

(\*Fonction qui prend en argument une liste de liste de couple l et un couple (Variable, valeur) et ajoute ce couple dans toutes les sous listes de l \*)

let rec ajoutercouple l (a,b) = match l with

[]->[]

|x::ll1 -> ((a,b)::x)::(ajoutercouple ll1 (a,b)) ;;

(\*Fonction finale question 2 qui prend en argument une liste de variable et renvoie une liste d'environnement pour e \*)

let rec listeenv e = match e with

[]-> [[]]

|[V(a)] -> [[(V(a),VRAI)] ; [(V(a),FAUX)]]

|x::ll1 -> let l1 ,l2 = listeenv ll1 , listeenv ll1

in concat (ajoutercouple l1 (x,VRAI))

(ajoutercouple l2 (x,FAUX));;

(\*Fonction qui prend un élément e de type eb et l’évalue \*)

let rec equa e = match e with

V(a) -> V(a)

|VRAI -> VRAI

|FAUX -> FAUX

|NOT(a) -> if equa a = VRAI then FAUX

else VRAI

|AND(a,b) -> if equa a = VRAI && equa b = VRAI then VRAI

else FAUX

|OR(a,b) -> if equa a = FAUX && equa b = FAUX then FAUX

else VRAI

|XOR(a,b) ->if equa a = equa b then FAUX

else VRAI

|NAND(a,b) -> if equa a = VRAI && equa b = VRAI then FAUX

else VRAI

(\*Fonction qui prend une équation booléenne et l'évalue\*)

let verifega e = match e with Equals(a,b) -> if equa a = equa b then true else false ;;

(\*Fonction qui prend un système d’équation sans inconnue,l’évalue et renvoie une liste de booléen [a0,...,an] ou chaque ai correspond à l'évaluation de la ième équation dans le système \*)

let rec verifens e = match e with

[]->[]

|x::ll1 -> (verifega x)::(verifens ll1);;

(\*Fonction qui prend une liste de booléen et renvoie true si tout les élément sont true false sinon\*)

let rec veriffinal e =match e with

[]->true

|x::ll1 -> if x = true then veriffinal ll1 else false;;

(\*Fonction qui prend un élément eb et un couple (variable , valeur) et remplace toutes les occurrences de variable dans eb par sa valeur \*)

let rec remplacer eb (var,vale) = match eb with

V(a) ->if V(a) = var then vale else V(a)

|VRAI -> VRAI

|FAUX -> FAUX

|NOT(a) -> NOT(remplacer a (var,vale))

|AND(a,b) ->AND(remplacer a (var,vale),remplacer b (var,vale))

|OR(a,b) ->OR(remplacer a (var,vale),remplacer b (var,vale))

|XOR(a,b) ->XOR(remplacer a (var,vale),remplacer b (var,vale))

|NAND(a,b) ->NAND(remplacer a (var,vale),remplacer b (var,vale));;

(\*Fonction qui prend une liste d'équation de type eb et un couple (variable, valeur) et remplace toutes les occurences de la variable par valeur dans le système syseq\*)

let rec remplacer2 syseq (var,vale) = match syseq with

[]->[]

|Equals(a,b)::ll1->Equals(remplacer a (var,vale),remplacer b (var,vale))::(remplacer2 ll1 (var,vale));;

(\*Fonction qui prend une liste d’équation et une liste de couple (var,val) et remplace toutes les variables par leur valeur dans le système d'équation \*)

let rec remplacer3 syseq l = match l with

[]->syseq

|x::ll1 ->remplacer3 (remplacer2 syseq x) ll1 ;;

(\*Fonction qui prend un système d’équation et une liste d'environnement et renvoie une liste de toutes les solutions possibles \*)

let rec f lens lenv = match lenv with

[]->[]

|x::ll1 -> if veriffinal (verifens (remplacer3 lens x))= true

then x::f lens ll1 else f lens ll1;;

(\*Fonnction finale\*)

let solveur e = f e (listeenv (listevar e)) ;;