INFO0054 - Projet : Tableaux Sémantiques

Alexandre Andries, Thomas Rotheudt, Abdelilah Khaliphi

Table des matières

1 Implémentation des formules, tableaux et extensions

La représentation des ensembles finis de formules repose sur les listes. Un ensemble de formules est une liste de formules, ces dernières étant à leur tour représentées sous forme de listes. En pratique, Racket représente donc un ensemble de formules comme une liste de listes. Le contenu des formules consiste en des "*quoted values*" (eg. '(OR a b)) est une formule valide).

Les formules sont en fait des formules propositionnelles contenant des variables propositionnelles (eg. a, b, etc.) et des opérateurs logiques acceptant chacun deux arguments, les arguments pouvant eux-mêmes comprendre des opérations (eg. '(AND a (OR b c))).

De même, les tableaux résultants des opérations prennent une forme similaire et sont représentés comme une liste de sous-tableau (étant eux-mêmes des listes). Ces tableaux ne contiennent en revanche aucun opérateur. Leurs éléments ne peuvent être que des variables propositionnelles, ou leur négation, représentée par la variable précédée de NOT (eg. '(NOT a)).

Un autre aspect fondamental de l'implémentation repose sur l'abstraction des données afin d'améliorer la lisibilité du code. Un fichier spécifique dédié à cette abstraction a été conçu, ses fonctions permettant respectivement de créer une liste de tableau, ajouter un tableau à une liste de tableau, créer la négation d'une variable, et récupérer la variable d'une négation. Ces fonctions sont représentées ci-dessous en langage Scheme.

```
1 (define cree-liste-tab list)
2 ; ------------------------------------ ;
3 (define ajout-tab list)
4 ; ------------------------------------ ;
5 (define (mk-NOT x) (list 'NOT x))
6 ; ------------------------------------ ;
7 (define get-proposition-NOT cadr)
```

2 Implémentation des règles d'éliminations

Le coeur de la fonction *semtab* se base sur l'élimination des opérateurs présents dans les formules de l'ensemble des formules. Il est donc nécessaire de mettre en place des règles afin de procéder aux éliminations successives.

Les trois opérations possibles actuellement dans une formule sont $\wedge, \vee, \implies, \equiv$ représentées respectivement par AND, OR, IFTHEN et EQUIV. Par extension, des règles pour XOR, XNOR, et NAND sont également définies. Par exemple, la formule $(a \wedge (b \vee c)) \implies a$ est traduite comme '(IFTHEN (AND a (OR b c)) a). La fonction *elim-operation* va traduire ces opérateurs selon la liste de règles reprise ci-dessous :

FIGURE 1 – Règles d'éliminations de l'opérateur OR

FIGURE 2 – Règles d'éliminations de l'opérateur AND

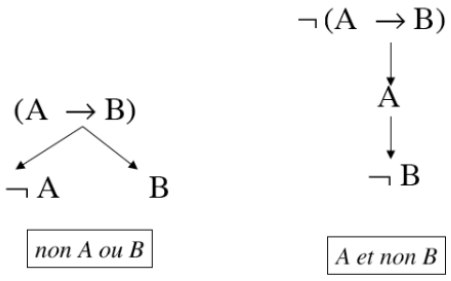

FIGURE 3 – Règles d'éliminations de l'opérateur IFTHEN

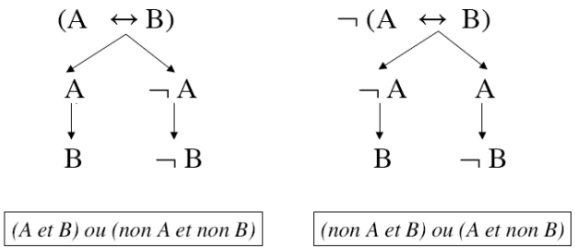

FIGURE 4 – Règles d'éliminations de l'opérateur EQUIV

L'implémentation se base sur la fonctionnalité *match* proposée par le language SCHEME. On obtient donc la fonction suivante :

```
; Élimine les opérations dites simples en un ou deux tableaux maximum
; Précondition: operation != null
;
; Retourne:
; liste contenant les différents tableaux d'opération
; (1 ou 2 tableau dans la liste en fonction des cas)
(define (elim-operation operation)
  (match operation
        ((list 'AND a b) (cree-liste-tab (ajout-tab a b)))
        ((list 'OR a b) (cree-liste-tab (ajout-tab a) (ajout-tab b)))
        ((list 'IFTHEN a b) (cree-liste-tab (ajout-tab (mk-NOT a))
                                            (ajout-tab b)))
        ((list 'NAND a b) (cree-liste-tab (ajout-tab (mk-NOT a))
                                          (ajout-tab (mk-NOT b))))
        ((list 'EQUIV a b) (cree-liste-tab (ajout-tab a b)
                                           (ajout-tab (mk-NOT a)
                                                      (mk-NOT b))))
        ((list 'XOR a b) (cree-liste-tab (ajout-tab a (mk-NOT b))
                                         (ajout-tab (mk-NOT a) b)))
        ((list 'XNOR a b) (cree-liste-tab (ajout-tab (mk-NOT a)
                                                     (mk-NOT b))
                                          (ajout-tab a b)))
        ((list 'NOT (list 'AND a b)) (cree-liste-tab (ajout-tab (mk-NOT a))
                                                     (ajout-tab (mk-NOT b))))
        ((list 'NOT (list 'OR a b)) (cree-liste-tab (ajout-tab (mk-NOT a)
                                                               (mk-NOT b))))
        ((list 'NOT (list 'IFTHEN a b)) (cree-liste-tab (ajout-tab a
                                                               (mk-NOT b))))
        ((list 'NOT (list 'NOT a)) (cree-liste-tab (ajout-tab a)))
        ((list 'NOT (list 'EQUIV a b)) (cree-liste-tab (ajout-tab (mk-NOT a) b)
                                                       (ajout-tab a (mk-NOT b))))
        ((list 'NOT a) (cree-liste-tab (ajout-tab(mk-NOT a))))


        (a (cree-liste-tab (ajout-tab a)))
    )
)
```

3 Architecture du code & Fonctions importantes

3.1 TABLEAU.RKT

Comme expliqué précédemment, le fichier TABLEAU.RKT reprend les fonctions et procédures permettant la gestion des types abstraits *formule* et *tableau*. Le fichier contient également les fonctions auxiliaires servant à la construction de *semtab*.

3.2 SEMTAB.RKT

Le fichier SEMTAB.RKT contient la fonction *semtab*. Celle-ci sert ensuite de fondation aux fonctions contenues dans le module (MODULE.RKT). Cette fonction est sans aucun doute la plus importante du projet car elle sert de pilier central aux opérations sur les listes de tableaux qui seront effectuées par la suite dans le fichier MODULE.RKT.

La fonction *semtab* consiste en l'élimination systématique des opérateurs trouvés dans les formules de la liste de formules. Pour effectuer cette élimination, la fonction appelle successivement une sous-fonction *cherche-elimination* afin de transformer chaque opérateur trouvé et ses arguments en tableaux correspondants aux traductions reprises ci-dessus dans les règles d'élimination. L'intérêt de la fonction est d'éviter une sur-consommation de la mémoire en cherchant premièrement à traduire les opérations ne créant pas de sous-branches (et donc de nouveaux sous-tableaux). Une fois ceux-ci éliminés, la fonction passe ensuite à la création des sous-tableaux. Dans le cas où il ne reste aucun opérateur à éliminer, la fonction renvoie *#f*. Ceci permet donc de vérifier si le tableau est ouvert ou fermé. La fonction est construite autour d'un accumulateur afin d'éviter une remontée récursive coûteuse. La forme itérative de *semtab* permet ainsi de meilleures performances.

A l'aide d'une fonction auxiliaire contenant l'algorithme et l'accumulateur et à partir d'une liste de formules de base F, la fonction va récursivement effectuer la conversion des éléments de F. La récursivité va procéder comme suit :

- ☐ CAS DE BASE : On vérifie si la liste de tableaux/formules est vide, ou non (i.e. l'ensemble des tableaux ont été traité, (ouvert - fermé))
- ☐ CAS RÉCURSIF : On cherche une opération à éliminer par la méthode décrite précédemment dans le premier tableau de la liste, on a donc deux cas possibles :
  1. Il est nécessaire de mettre à jour le reste de la liste de tableaux en ajoutant les tableaux renvoyés par l'élimination trouvée, concaténés au reste du tableau en cours d'analyse.
  2. Le tableau en cours d'analyse est ajouté à l'accumulateur car il est déjà traité et il n'y a plus de modification à effectuer. On procède ensuite à la récursion sur la suite du tableau de formules.

3.3 MODULE.RKT

Le fichier SEMTAB.RKT contient quant à lui toutes les fonctions permettant d'analyser le résultat renvoyé par *semtab*. Ces fonctions sont décrites dans les sous-sections ci-dessous.

3.3.1 satisfiable ?

La fonction *satisfiable ?* détermine si l'ensemble de formules F est satisfaisable, ou non. La fonction vérifie s'il existe au moins un tableau vrai parmi tous les tableaux renvoyés par *semtab* appliqué à F.

3.3.2 valid ?

valid ? détermine si une formule f est valide sous la liste de formules F. Il existe deux cas. La premier cas consiste à vérifier si les modèles de f sont égaux aux modèles de F. Le deuxième cas consiste à vérifier si l'union de la liste de formulesµ avec f est satisfaisable ou non.

3.3.3 tautology ?

La fonction vérifie si une formule est une tautologie, c'est-à-dire qu'elle est toujours vraie.

3.3.4 contradiction ?

A l'inverse de *tautology ?*, la fonction vérifie si la formule est toujours fausse.

3.3.5 models

Cette fonction permet de récupérer la liste des tableaux issus de *semtab* et d'en tirer les modèles. C'est-à-dire, la liste de tous les tableaux contenant tous les éléments de la liste de formules de base F, qui ne sont pas des contradictions.

3.3.6 counterexamples

La fonction permet de vérifier d'abord si l'union d'une liste de formule F et la négation d'une formule f est satisfaisable. Si c'est le cas, elle affiche ensuite les contre-exemples.

4 Exemples d'utilisation des modules

Ci-dessous sont repris des exemples d'exécutions des fonctions décrites dans le rapport.

4.1 semtab

```
(semtab '((IFTHEN p (IFTHEN q r)) (NOT (IFTHEN (IFTHEN p q) r))))
```

On obtient :

4.2 tautology ?

```
(define test-tautology '(OR a (NOT a)))
(tautology? test-tautology)
```

On obtient : $\#t$

4.3 satisfiable ?

```
(define test-satisfiable '(a (NOT b) (IFTHEN (NOT b) c)))
(satisfiable? test-satisfiable)
```

On obtient : $\#t$

4.4 contradiction ?

```
(define test-contradiction '(AND a (OR b (NOT a))))
(contradiction? test-contradiction)
```

On obtient : $\#t$

4.5 counterexamples

```
(counterexamples 'a '((OR a (NOT a))))
```

On obtient : '(((NOT a)))

4.6 models

```
(define test-models '((IFTHEN p (IFTHEN q r)) (NOT (IFTHEN (IFTHEN p q) r))))
(models test-models)
```

On obtient : '(((NOT p) (NOT r) (NOT q)) ((NOT p) (NOT r) q))

Bibliographie

[Gri00] Pascal GRIBOMONT. *Eléments de programmation en Scheme*. Dunod, 2000.

[Ste21] James W. STELLY. *Racket Programming the Fun Way*. No Starch Press, 2021.

[Koe] Georges KOEPFLER. "Arbres de Beth". In : *Numération et Logique* (), p. 250-270.

[Mat] PLT MATTHEW FLATT Robert Bruce Findler. *The Racket Guide*. URL : `https://docs.racket-lang.org/guide/index.html`. (accessed : 19.12.2021).