UNIVERSITÉ DE ROUEN NORMANDIE

Projet de raisonnement propositionnel

Edouard.H Théo.R.V

2022–2023

Résumé

Ce document constitue notre compte rendu du projet de raisonnement propositionnel traite d'une implémentation de la méthode des tableaux et de fonctions de test. Il s'agit d'une méthode permettant de déterminer la satisfaisabilité d'une formule booléenne. Les fonctions de test permettent de vérifier empiriquement l'implémentation de la méthode.

Dans un premier temps, nous présenterons notre implémentation de la méthode des tableaux et des divers choix présents dans celle-ci. Pour finir, nous aborderons les difficultés liées à la production de ce projet.

Table des matières

I Implémentation de la méthode des tableaux

1 Formule

Tout d'abord, nous avons dès le début de la conception de ce projet mis en place les opérateurs étendus vus en TD. On peut notamment citées les opérateurs, **xor**, **nor**, **nand**, **diff** et **equiv** de symboles respectifs $\oplus$, $\uparrow$, $\downarrow$, $\setminus$ et $\Leftrightarrow$.

La fonction **string_of_formule** est une fonction récursive non terminale, ce choix est dû à plusieurs raisons. La première étant que sa complexité espace serait logarithmique par la division de la formule par deux à chaque appel récursif. La deuxième raison réside dans l'hypothèse que les formules affichées ne devraient pas être d'une taille démesurée, une formule trop grande ne pourrait pas être affichée sur un terminal de taille standard. De plus ce choix favorise la compréhension de cette fonction.

En ce qui concerne la fonction **eval** ayant déjà implémenté la partie de 'base' en TP avec les opérateurs **Ou**, **Et**, **Atome**, **Imp**, **Bot**, **Top** et **Non**, nous allons donc se focaliser sur les opérateurs étendus. Ces opérations n'étant pas implémentées de façon standard en Ocaml, nous avons donc à partir des tables de vérités misent en place des formules équivalentes permettant alors leur implémentation, voir table 1 pour les formules équivalentes résultant des tables de vérités.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0
$(a \setminus b) \vee (b \setminus a)$		

a	b	$a \uparrow b$
0	0	1
0	1	1
1	0	1
1	1	0
$\neg(a \wedge b)$		

a	b	$a \downarrow b$
0	0	1
0	1	0
1	0	0
1	1	0
$\neg(a \vee b)$		

a	b	$a \setminus b$
0	0	0
0	1	1
1	0	1
1	1	0
$a \vee b$		

a	b	$a \Leftrightarrow b$
0	0	0
0	1	1
1	0	1
1	1	0
$(a \rightarrow b) \wedge (b \rightarrow a)$		

TABLE 1 – Tables de vérités et formules équivalentes.

2 RandomFormule

Pour la fonction **random_form**, nous avons fait le choix de la divisée en plusieurs sous-fonctions. Un tel choix permet une compréhension plus aisée du traitement effectué dans la fonction. Les quatre fonctions **random_atome**, **random_n_operator**, **random_u_operator** et **random_b_operator** correspondent chacune à l'obtention d'un élément de façon pseudo-aléatoire, un atome, un opérateur nullaire, unaire et binaire respectivement aux fonctions citées plus haut. Une telle division permet aussi de facilitée un possible futur ajout d'opérateurs, d'où la création d'une fonction pour obtenir un opérateur unaire de façon aléatoire bien qu'il n'aille qu'une seule possibilité. Toutes les autres fonctions obtiennent l'élément cible en tirant un nombre pseudo-aléatoire à l'aide du module **Random**. L'algorithme utilisé pour la génération de cette formule pseudo-aléatoire est celui donné dans le sujet du projet.

3 Tableaux

Toutes les fonctions implémentées dans le fichier source **Tableaux.ml**, résulte de l'implémentation de la méthode des tableaux.

Pour les fonctions **tableau_sat** et **tableau_ex_sat**, nous avons réunie ces traitements en une seule fonction auxiliaire **tab_methode**. Elle renvoie un couple de booléen et d'une liste d'atome, liste prouvant sa satisfaisabilité dans le cas ou la formule **f** l'est.

C'est une fonction récursive non-terminale (car supposant que les formules sont de tailles raisonnables), qui itère sur la liste de formule.

En effet, la méthode des tableaux décompose une formule pour construire un arbre permettant de déterminer la satisfaisabilité de cette formule. L'itération sur la liste de formules permet comme décrit dans la méthode d'ajouter en début de 'branche' les formules à traiter par la suite.

Comme décrit dans cette méthode, l'opérateur **Bot** conclue à une fermeture de branche, c'est-à-dire que cette branche ne peut pas être satisfaisable, d'où le retour pour cette branche de la valeur **false** et d'une liste d'atomes vides.

Pour le cas de **Top**, la fonction passe à la prochaine sous-formule à traiter en conservant les atomes rencontrés précédemment. En cas de rencontre d'un atome qu'il s'agisse de sa négation ou non, une recherche dans la liste d'atomes déjà rencontrée permet de savoir si sa négation est présente, dans ce cas la branche est fermée (donc de même retour que pour l'opérateur **Bot**) sinon l'ajoute à la liste d'atomes puis effectue le prochain traitement.

La rencontre d'une conjonction et d'une négation de disjonction, entraine l'ajout en tête de liste des deux opérandes et de leurs négations respectivement et continue le traitement.

En ce qui concerne la disjonction et la négation de conjonction, nous créons un 'nœud' à la fin de la branche. Ce 'nœud' est représenté par une première exécution de la branche avec en tête de celle-ci. Si cette exécution renvoie vrai alors la formule est satisfaisable donc on arrête le traitement sinon le programme exécute une nouvelle fois cette branche, mais avec en tête cette fois-ci la deuxième opérande.

Pour finir en cas d'apparition d'un opérateur étendu, un appel à la fonction **transform** permet d'exprimer cette opération en fonction des opérateurs dont le traitement est cité ci-dessus.

En ce qui concerne la fonction **tableau_all_sat**, le traitement de tous les opérateurs est identique à celui de la fonction **tab_methode** sauf pour la disjonction. En effet, devant renvoyer touts les atomes de toutes les branches, de ce fait toutes les branches sont traité pour les disjonctions. Puis tous les résultats des branches sont mis dans une liste de liste d'atomes

4 Test

La fonction **to_alea_inter**, a été implémentée à l'aide d'une fonction auxiliaire qui sera renvoyée à l'utilisateur partiellement exécuté (sans tous ses paramètres, pour renvoyer une fonction de type interprétation). On peut remarquer l'utilisation de la fonction **List.assoc_opt**, qui permet de rechercher dans une liste de couple clé valeur (a, b) la valeur de b qui correspond à la valeur de clé passée en paramètres. On remarque quelle renvoie **None** en cas d'échec et **Some b** en cas succès. L'interprétation renvoyée par cette fonction comble les atomes qui ne sont pas présent dans la liste **sbl** (liste de couple booléen string) par un booléen choisi de façon pseudo-aléatoire.

La fonction permettant de tester empiriquement notre implémentation de la méthode des tableaux est **test_valid**. Pour ce test, elle va générer une formule pseudo-aléatoire, à l'aide de la fonction **random_form** (décrite dans la sous-section 2) de **n** opérateurs et d'atomes possible la liste 'a', 'b', 'c' et 'd'. Ensuite elle va exécuter les fonctions **ex_sat** et **all_sat** qui correspondent à un appel à la fonction **eval** sur les listes d'atomes en résultat des fonctions **tableau_ex_sat** et **tableau_all_sat**. La fonction **eval** utilise l'interprétation renvoyée par la fonction **to_alea_inter**.

II Les difficultés rencontrées

Tous d'abord, notre première difficulté a été la simplification des opérateurs en fonction des opérateurs de 'base'. Nous avons dû réaliser des tables de vérités, a fin de trouver les opérations correspondantes (voir tableau 1).

Une autre difficulté a été une tentative de simplification, mais aussi de factorisation des codes des fonctions **tab_methode** et **tableau_all_sat**. En effet, on voit très clairement que les deux fonctions ont un code très similaire. Mais malgré ces diverses tentatives aucune amélioration du code n'a pu être trouvée.

Pour finir, notre plus grosse difficulté qui ne résulte pas forcément de ce projet, a été la compréhension de l'énonce, mais aussi la production de ce rapport.

III Conclusion

Pour conclure, nous avons implémenté cette méthode sans trop de difficultés après l'avoir compris. Nous l'avons testé de nombreuses fois à l'aide de la fonction **test_valid**. Elle reste plus efficace que l'exécution d'un algorithme calculant la table de vérité d'une formule dans la plupart des cas. Malgré cela, il reste de nombreuses améliorations à produire notamment sur la factorisation du code.

1 Remerciement

Merci à, Erwan LIEVIN, pour avoir effectué une relecture des fautes d'orthographes présentes dans ce compte-rendu.