Projet de raisonnement propositionnel

Edouard.H Théo.R.V

2022–2023

Résumé

Ce document constitue notre compte rendu du projet de raisonnement propositionnel traitent d'une implémentation de la méthode des tableaux et de fonction de test. Il s'agit d'une méthode permettant de déterminer la satisfesabiliter 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 présent 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 citée les opérateurs, **xor**, **nor**, **nand**, **diff** et **equiv** de symbole respectif $\oplus$, $\uparrow$, $\downarrow$, $\setminus$ 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 sera logarithmique par la division de la formule par deux a chaque appel récursif. La deuxième raison réside dans l'hypothèse que les formule afficher ne devrais pas être d'une taille démesurée enfaite 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 tout le travail, c'est focaliser sur les opérateurs étendus. Ces opérations n'étant pas implémenté de façon standard en Ocaml, nous avons donc à partir des tables de vérités mises en place des formules équivalente permettant alors leur implémentation, voir table 1 pour les formules équivalente 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 formule équivalente.

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** corresponde 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 facilité 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'un seul. 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 réunies ces traitement en une seule fonction auxiliaire **tab_methode**. Elle renvoie un couple de booléen et d'une liste, d'atome list prouvant sa satisfaisabilité dans le cas ou la formule f l'est. C'est une fonction récursive non-terminal (car supposant que les formules ont des tailles raisonnable), qui itère sur la liste de formule qu'il lui reste à traiter. 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 formule permet comme décrite dans la méthode d'ajouter en bout de 'branche' les formules à traiter par la suite. Comme décris 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 retours pour cette branche de la valeur **false** et d'une liste d'atomes vide. Pour le cas de **Top**, la fonction passe à la prochaine sous-formule à traiter en conservant les atomes rencontrée précédemment. En cas de rencontre d'un atome qu'il sagise de sa négation ou non, une recherche dans la liste d'atome déjà rencontrée permet de savoir si sa négation est présente, dans ce cas la branche est fermé (donc de même retours que pour l'operateur **bot**) sinon l'ajoute à la liste d'atome puis éffectue le prochain traitement. La rencontre d'une conjonction et d'une négation de disjonction, entraine l'ajout en queue de liste des deux opérandes et de leurs négation respectivement et continue le traitement. Cette ajout est mis en place par une concaténation de liste, pour permettre facilement cette ajout en queue. En ce qui concerne la disjonction et la négation de conjonction, nous créons un 'noeud' à la fin de la branche,

4 Test

La fonction **to_alea_inter**, a été implémentée à l'aide d'une fonction auxiliaire qui sera renvoyer à l'utilisateur partiellement exécuter (sans tous ces 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 corresponds à la valeur de clé passer en paramètres. On remarque quelle renvoie **None** en cas d'échec et **Some b** en cas succès.

Pour la fonction **test_valid**, nous utilisont des sous-fonctions **ex_sat**, **all_sat**. Chacune visent à vérifier la véracité des retours des fonctions **tableau_ex_sat** et **tableau_all_sat** respectivement.

II Les difficultés rencontrées

III Conclusion