Rapport projet d'ORP

Tristan GROULT

16 mai 2025

Sommaire

1 Introduction

La transformation d'une formule en une forme clausale conjonctive, interprétée comme une conjonction de disjonctions de littéraux, est une étape nécessaire pour l'application de l'algorithme DPLL permettant de décider de la satisfaisabilité d'une formule.

Il est possible d'adapter l'algorithme pour l'utiliser sur des formes clausales disjonctives menant à un algorithme que nous appellerons coDPLL.

Le but de ce projet est d'étudier la mise en forme clausale disjonctive et de proposer une modification adaptée depuis l'algorithme DPLL, puis de tester les résultats obtenus à l'aide d'un générateur aléatoire de formules.

Nous considèrerons des formules de la logique propositionnelle utilisant les opérateurs classiques (négation, conjonction, disjonction, implication, vrai et faux) ainsi que l'opérateur d'équivalence.

2 Forme clausale

Une forme clausale disjonctive est évaluée comme la disjonction des clauses qu'elle contient, et une clause est évaluée comme la conjonction des littéraux qu'elle contient.

Par exemple, la forme clausale disjonctive $\{\{a, b\}, \{\neg a, \neg c\}, \{\neg c, \neg b\}, \{c\}\}$ est constituée de quatre clauses et est évaluée comme la formule $(a \wedge b) \vee (\neg a \wedge \neg c) \vee (\neg c \wedge \neg b) \vee c$.

— la forme clausale vide est interprétée comme l'élément neutre de la disjonction, c'est-à-dire $\top$

— la clause vide est interprétée comme l'élément neutre de la conjonction, c'est-à-dire $\bot$

La mise en forme clausale suit les mêmes étapes pour les formes clausales conjonctives et pour les formes clausales disjonctives.

Le *retrait des négations* et *la descente des négations* étant les mêmes car ne dépendant pas de la forme ensembliste finale, les fonctions `retrait_operateurs` et `descente_non` sont placées dans le même fichier `FC.ml`, global pour la gestion des formes clausales.

L'étape qui change est la *mise en forme ensembliste*. C'est à cette étape que la forme clausale est formée. Deux fonctions sont donc nécessaires pour gérer les deux mises en forme ensemblistes. La nouvelle fonction `forme_ensembliste` pour la forme clausale disjonctive, calcule alors inductivement :

— les atomes niés $\neg a$ sont transformés en forme clausale contenant la clause réduite au littéral négatif $\{\{\neg a\}\}$;

— les atomes non niés a sont transformés en forme clausale contenant la clause réduite au littéral positif $\{\{a\}\}$

— l'opérateur $\bot$ est remplacé par la forme clausale vide $\varnothing$;

— l'opérateur $\top$ est remplacé par la forme clausale contenant la clause vide $\{\varnothing\}$;

— la disjonction de deux formes clausales $\{C_1, \ldots, C_n\}$ et $\{C'_1, \ldots, C'_m\}$ correspond à leur union, c'est-à-dire $\{C_1, \ldots, C_n, C'_1, \ldots, C'_m\}$ par associativité, commutativité et idempotence de la disjonction ;

— la conjonction de deux formes clausales $\{C_1, \ldots, C_n\}$ et $\{C'_1, \ldots, C'_m\}$ correspond à la combinaison cartésienne de leurs clauses, c'est-à-dire $\{C_1 \cup C'_1, \ldots, C_1 \cup C'_m, \ldots, C_n \cup C'_1, \ldots, C_n \cup C'_m\}$ par distributivité de $\wedge$ sur $\vee$;

Les fonctions `fcd_to_formule` et `fcc_to_formule` parcourent la forme clausale sur laquelle elles sont appelées en réalisant :

— La disjonction des clauses de conjonction des littéraux pour la forme clausale disjonctive
— La conjonction des clauses de disjonction des littéraux pour la forme clausale conjonctive

3 Algorithme coDPLL

L'algorithme basique pour coDPLL est basé sur l'algorithme basique de DPLL. Pour appliquer cet algorithme, il faut savoir déterminer la simplification d'une forme clausale disjonctive $\mathcal{F}$ si on considère un littéral l comme vrai (et donc l' à faux). Cette forme simplifiée s'obtient donc en éliminant de $\mathcal{F}$ les clauses contenant l' et en éliminant l des clauses restantes.

Algorithme basique coDPLL

Teste la satisfaisabilité d'une forme clausale disjonctive $\mathcal{F}$

1. Si $\mathcal{F}$ est vide, renvoyer `faux`
2. Sinon, si la clause vide appartient à $\mathcal{F}$, renvoyer `vrai`
3. Sinon, choisir une clause C de $\mathcal{F}$ et un littéral l de C. En considérant l' le littéral complémentaire de l :
   (a) Simplifier $\mathcal{F}$ en considérant l'hypothèse que l est vrai (et donc que l' est faux), ce qui donne $\mathcal{F}'$. Si $\mathcal{F}'$ est satisfaisable (déterminé récursivement), renvoie `vrai`.
   (b) Simplifier $\mathcal{F}$ en considérant l'hypothèse que l est faux (et donc que l' est vrai), ce qui donne $\mathcal{F}''$. Si $\mathcal{F}''$ est satisfaisable (déterminé récursivement), renvoie `vrai` sinon `faux`.

4 Module Test

Le module test, grâce à la fonction `test_log` permet un affichage de différents tests nous permettant de vérifier la validité de nos implantations.

Pour cela, nous avons dû développer les fonctions `correct_ex_sat` et `correct_all_sat` permettant de vérifier la validité des fonctions d'application de l'algorithme DPLL renvoyant un/des témoin(s) pour que la formule soit satisfaisable.

Ces 2 fonctions réalisent donc l'évaluation de la fonction qui leur est passée en paramètres à partir d'une interprétation basée sur les témoins qui leur sont donnés afin de vérifier leur validité. Nous avons décidé ici d'utiliser la fonction `eval` déjà écrite précédemment en TP dans le module formule et d'inclure avec cela le type `interpretation` de forme `type interpretation = string -> bool`.

Ce type `interpretation` est utilisé dans plusieurs fonctions permettant la réalisation d'une fonction d'interprétation à partir des témoins passés en paramètres ou à partir de listes d'atomes d'une formule.

La vérification est ensuite faite en suivant les algorithmes définis dans la documentation.