Fiche 4

Projet : l'algorithme DPLL (simplifié).

1 Avant-propos

L'évaluation de TP, c'est-à-dire ce projet, est un examen universitaire comptant pour la délivrance du diplôme national de licence.

À ce titre, toute fraude ou plagiat entraine la rédaction d'un procès verbal à l'attention de la section disciplinaire de l'université, qui peut infliger, en plus de la nullité de l'épreuve (note de 0), une sanction allant de l'avertissement à l'exclusion définitive de tout établissement public d'enseignement supérieur.

Ainsi, si vous utilisez un morceau de code que vous n'avez pas produit, vous devrez indiquer très clairement sa source (site internet, camarade de promotion, *etc.*) : vous ne serez pas accusé de plagiat, mais ce morceau de code ne sera pas compté dans votre note finale.

2 Modalités de réalisation et de rendu

Le projet est à réaliser par groupes de **deux étudiants maximum**. Vous devrez rédiger un rapport de projet contenant :

1. les réponses aux questions de la Section 4,
2. les choix de programmation,
3. les difficultés rencontrées et les solutions apportées.

Vous devrez rendre le projet (rapport et code source dans une archive) pour le lundi 2 janvier au plus tard, en déposant celui-ci sur universitice dans le dépôt correspondant. Une soutenance de 15 minutes par binôme aura lieu le jeudi 5 ou le vendredi 6 janvier afin de vous interroger sur votre réalisation : vous serez interrogés au tableau sur l'application de l'algorithme détaillé dans la suite sur une formule donnée, puis devrez expliquer les différentes étapes de cet algorithme dans votre code.

3 Introduction

L'algorithme DPLL, nommé selon ses auteurs (Davis, Putnam, Logemann et Loveland), est une réinterprétation optimisée de l'algorithme de Quine sur des formes clausales conjonctives. Ainsi, au lieu de simplifier une formule après avoir choisi un atome, on simplifie une forme clausale après avoir choisi un littéral, positif ou négatif. Des optimisations sont utilisées dans la version classique de l'algorithme, telles que la propagation unitaire ou l'élimination des littéraux purs.

Le but de ce projet est d'implanter une version simplifiée de l'algorithme DPLL en OCaml et de l'utiliser pour étudier la satisfaisabilité de formules Booléennes de la logique propositionnelle. Vous devrez également mesurer l'efficacité de votre implantation par la résolution de Sudoku. Il ne vous est pas demandé d'implanter les optimisations de propagation unitaire ou d'élimination des littéraux purs.

4 Le choix d'un littéral et la simplification de forme clausale

Comme pour l'algorithme de Quine, la première étape est de choisir un littéral d'une clause de la forme clausale conjonctive étudiée. Pour cela, la forme clausale conjonctive doit contenir une clause non vide. Une fois le littéral choisi, des simplifications peuvent être réalisées en supposant que ce littéral soit vrai ou faux.

Question 1

Comment sont évaluées

1. la forme clausale conjonctive vide
2. et une forme clausale conjonctive contenant la clause vide ?

Question 2

Considérons une clause $C_1 = \{\ell_1, \ell_2, \ldots, \ell_n\}$ contenue dans une forme clausale conjonctive $\mathcal{F} = \{C_1, C_2, \ldots, C_n\}$. Expliquer comment simplifier $\mathcal{F}$ et chacune de ses clauses si le littéral ℓ_1

1. est supposé faux ?
2. est supposé vrai ?

Question 3

Sur le modèle de l'algorithme de Quine vu en cours, proposer une rédaction de l'algorithme DPLL simplifié permettant de déterminer si une forme clausale conjonctive est satisfaisable, en suivant le paradigme fonctionnel, sur la base d'une succession de la forme : choix de littéral — simplifications — appels récursifs correspondants.

5 Implantation : structure des ressources de départ

La structure du projet et l'utilisation de `dune` sont imposées.
La structure de départ est la suivante :

— un dossier `Proposition` contenant la bibliothèque de manipulation des formules et l'implantation de l'algorithme DPLL ;
— un dossier `Sudoku` contenant un exécutable permettant de mesurer l'efficacité de votre code sur la résolution de Sudoku ;
— un dossier `aux` contenant des fichiers d'aide de différents formats, détaillés dans la suite.

Vous devrez également considérer l'extraction de formules et de formes clausales depuis des fichiers d'un format particulier (plus précisément, depuis un format DIMACS simplifié). Ces fichiers textes sont constitués de lignes d'éléments séparés par des espaces, où chaque élément est de la forme *+atome* ou *-atome*. Un élément *+atome* correspond à la formule atomique atome et un élément *-atome* correspond à la formule ¬atome. Une ligne correspond à la disjonction des formules associées à ses éléments constitutifs. Un fichier correspond à la conjonction des formules associées à chacune de ses lignes. Vous trouverez un exemple de fichier dans le dossier aux. Celui-ci contient les lignes suivantes :

```
+x1 -x2 +x4 +x5
+x5 -x8 +x3
-x4 -x8 -x6
```

correspondant à la formule

$$(x_1 \vee \neg x_2 \vee x_4 \vee x_5) \wedge (x_5 \vee \neg x_8 \vee x_3) \wedge (\neg x_4 \vee \neg x_8 \vee \neg x_6).$$

5.1 Le dossier Proposition

Ce dossier contient la bibliothèque (nommée `Proposition`) contenant les différents modules pour l'implantation de l'algorithme DPLL.

Le fichier `Formule.ml` contient les éléments de manipulation de formules Booléennes dont le type est imposé. Vous devrez implanter les fonctions suivantes en respectant les signatures fournies :

— `string_of_formule` transformant une formule en chaine de caractères,
— des opérateurs de constructions de formules simplifiées : `( + )`, `( * )`, *etc.*

— `from_file` transformant un fichier au format DIMACS simplifié en une formule Booléenne.

Le fichier `FCC.ml` contient les éléments de manipulation des formes clausales conjonctives. Vous devrez implanter les fonctions suivantes en respectant les signatures fournies :

— `string_of_fcc` transformant une forme clausale en chaine de caractères ;
— la mise en forme clausale conjonctive d'une formule ;
— l'import depuis un fichier au format DIMACS simplifié.

Le fichier `DPLL.ml` contient l'implantation de l'algorithme DPLL. Vous devrez implanter les fonctions suivantes en respectant les signatures fournies :

— `simplif_fcc` simplifiant une forme clausale conjonctive en fonction d'un littéral, positif ou négatif, supposé vrai ;
— les différentes fonctions d'étude de satisfaisabilité d'une formule en utilisant l'algorithme DPLL.

Une fois ces éléments implantés, vous pourrez les tester à l'aide du fichier `Test.ml`, en y ayant décommenté les éléments correspondants puis en lançant la commande `dune utop` depuis le dossier racine du projet et en chargeant les modules nécessaires :

```
open Proposition;;
open Formule;;
open FCC;;
open DPLL;;
open Test;;
test_input_formule "aux/test.input";;
```

5.2 Le dossier Sudoku

Ce dossier contient les mêmes éléments que le solveur de Sudoku que vous avez utilisé lors de la troisième séance de TP. Il vous suffit de modifier le fichier `Adapteurs.ml` pour que la résolution de Sudoku puisse être utilisée. Vous pourrez lancer l'exécutable par la commande `dune exec ./Sudoku/Sudoku.exe` suivie des arguments choisis, lancée dans un terminal ouvert à la racine du projet.

6 Optimisations

Afin d'accélérer le traitement de l'algorithme DPLL, il vous est demandé d'implanter deux améliorations :

— les clauses contenues dans une forme clausale devront être classées militairement, c'est-à-dire selon leurs cardinaux puis par l'ordre du module `Clause`. Pour cela, vous devrez modifier la fonction de comparaison du module utilisé par le foncteur construisant le module des formes clausales ;
— les formules Tous(xs) et UnSeul(xs), ou leurs formules équivalentes, devront être transformées en des conjonctions de disjonctions de petites tailles. Ainsi, dans le fichier `Adapteurs.ml`, les fonctions devront considérer les éléments suivants :
  — la formule Tous(xs) peut être transformée en la conjonction des formules de xs ;
  — la formule UnSeul(xs) peut être transformée en une formule combinant conjonctivement la disjonction des éléments de xs et empêchant chaque couple de formules de xs d'être vraies en mêmes temps :

$$\begin{aligned}\text{UnSeul}(F_1, F_2, \ldots, F_k) =& (F_1 \vee \cdots \vee F_k)\\ &\wedge(\neg F_1 \vee \neg F_2) \wedge \cdots \wedge (\neg F_1 \vee \neg F_k)\\ &\wedge(\neg F_2 \vee \neg F_3) \wedge \cdots \wedge (\neg F_2 \vee \neg F_k)\\ &\wedge \cdots\\ &\wedge(\neg F_{k-1} \vee \neg F_k)\end{aligned}$$

7 Barème indicatif

1. Code (13 points) :

   (a) Module `Formule` (2 points) :
      — représentation en chaine de caractère (0.5 point)
      — opérateurs de construction (0.5 point)
      — import depuis un fichier DIMACS simplifié (1 point)
   (b) Module `FCC` (2 points) :
      — représentation en chaine de caractère (0.5 point)
      — mise en forme clausale d'une formule (0.5 point)
      — import depuis un fichier DIMACS simplifié (1 point)
   (c) Module `DPLL` (6 points) :
      — simplification d'une FCC (2 points)
      — études de la satisfaisabilité (4 points)
   (d) Optimisation (2 points)
   (e) Adaptateurs du Sudoku (1 point)
2. Rapport (4 points) :
   — Réponses aux questions de la Section 4 (2 points)
   — autres éléments attendus (2 points)
3. Soutenance (3 points)

Il s'agit d'un barème indicatif "de départ". À celui-ci s'ajouteront différentes pénalités, notamment si :
— votre code rencontre des erreurs de compilation ;
— votre code émet des avertissements à la compilation ;
— des contre-exemples sont détectés,

ainsi que des bonus, notamment si :
— votre rapport est écrit en LaTeX,
— vous codez d'une façon propre, lisible, élégante et efficace,
— vous intégrez les optimisations de l'algorithme DPLL classique (propagation unitaire ou élimination des littéraux purs) dans une seconde version de votre implantation, en conservant une version ne contenant pas ces optimisations. Dans ce cas, vous devrez indiquer clairement les sources étudiées pour appliquer ces optimisations.