Projet IA303
2019-2020

Alexandre Chapoutot

3 décembre 2019

L'objectif du projet est la mise en œuvre d'un algorithme de satisfiabilité de formules logiques propositionnelles fondés sur la méthode DPLL.

Nous utiliserons le langage Python (version 3) pour ce projet. Nous nous appuierons la bibliothèque `boolean.py`[1] de Python pour faciliter la lecture d'une chaîne de caractères définissant une formule de logique propositionnelle.

L'installation de cette bibliothèque s'effectue de la manière suivante

```
pip install boolean.py
```

Cette formule sera composée des opérateurs AND (&), OR (|) et de la négation (!) ainsi que de symboles. Par exemple,

```
>>> import boolean
>>> algebra = boolean.BooleanAlgebra()
>>> algebra.parse('x & y')
AND(Symbol('x'), Symbol('y'))
```

En utilisant les attributs `operator` et `args` vous pouvez visiter les formules logiques

```
>>> algebra.parse('x | !y ').operator
'|'
>>> algebra.parse('x | !y ').args
(Symbol('x'), NOT(Symbol('y')))
```

Au besoin vous pouvez convertir une formule logique au format CNF

```
>>> toto = algebra.parse('x | !y & (z | t)  ')
>>> algebra.cnf (toto)
AND(OR(Symbol('t'), Symbol('x'), Symbol('z')), OR(Symbol('x'), NOT(Symbol('y'))))
```

1 Premier travail

En vous appuyant sur la bibliothèque `boolean.py` mettez en œuvre un programme avec les éléments suivants

- Lire une formule logique sous forme d'une chaîne de caractères
- Transformer cette formule en CNF (à l'aide de la fonction `cnf()` pour le moment)
- Traduire la formule CNF dans le format DIMACS
- Coder l'algorithme DPLL vu en cours pour résoudre la formule logique donnée dans le format DIMACS

1. `https://booleanpy.readthedocs.io/en/latest/index.html`

Pour mémoire, la méthode DPLL est définie par

Require: CNF formula F, empty model M
Ensure: UNSAT or SAT with model M
  procedure DPLL(F, M)
    $(F, M) \leftarrow$ UnitPropagate(F, M)
    if All clauses are true in M **then**
      return `SAT`
    end if
    if One clause is wrong in M **then**
      return `UNSAT`
    end if

    $\ell \leftarrow$ choose a literal not assigned in M
    if DPLL$(F_\ell, M \cup \{\ell\}) =$ SAT **then**
      return `SAT`
    end if
     return DPLL$(F_{\neg\ell}, M \cup \{\neg\ell\})$
  end procedure

2 Extension : transformation de Tseitin

Au lieu d'utiliser la fonction `cnf()` mettez en œuvre la transformation de Tseitin.

La transformation de Tseitin [2] permet d'obtenir à partir d'une formule f une CNF équisatisfiable (est satisfiable si et seulement si f l'est). Cette transformation est définie par une fonction récursive `tseitin`, qui ajoute des clauses générées à une CNF passée en argument. Son principe de fonctionnement est le suivant : étant donnée f, la fonction renvoie un nouveau littéral ℓ ainsi qu'une CNF c tels que c et ℓ s'évaluent à vrai si et seulement si f s'évalue à vrai. La fonction `tseitin(f)` est définie par les règles

- `tseitin(p) = (p,vide)`
- `tseitin(!f)` : soit `(p',c') = tseitin(f)` alors renvoyer `(!p',c')`
- `tseitin(f1 | f2)` : soient `(p1,c1) = tseitin(f1)`, `(p2,c2) = tseitin(f2)` et p un nouveau littéral renvoyer `(p, (!p | p1 | p2) & (p | !p1) & (p | !p2) & c1 & c2)`
- `tseitin(f1 & f2)` : soient `(p1,c1) = tseitin(f1)`, `(p2,c2) = tseitin(f2)` et p un nouveau littéral renvoyer `(p, (p | !p1 | !p2) & (!p | p1) & (!p | p2) & c1 & c2)`

La CNF équisatifiable finale est obtenue de la manière suivante :

```
soit (l,c) = tseitin(f)
la CNF est l & c
```

3 Extension (Optionnelle) : Two watched literals

Mettez en œuvre la méthode Two Watched Litterals pour détecter rapidement les clauses unitaires.

2. `https://en.wikipedia.org/wiki/Tseytin_transformation`