Rapport sur TP noté (résolution de 2-SAT)

Wassim Saidane, Aurélien Authier

Table des matières

1 Nombre maximum de clause (Exercice 1)

Soit n le nombre de variable et p le nombre de variables dans une clause (ici 2).

Prenons $n = 2$,
On a au maximum 4 variables en prenant en compte les complémentaires, pour trouver le maximum on utilise la combinatoire on a donc 2 parmis 4.
On a donc :
$\frac{4!}{2!\times 2!} = \frac{24}{4} = 6$

Prenons $n = 3$
On a au maximum 6 variables et de la même façon on obitent :
$\frac{6!}{2!\times 4!} = \frac{720}{2\times 24} = \frac{720}{48} = 15$

La formule pour déterminer le maximum de clause est donc :

$$\binom{2n}{p} = \left(\frac{2n!}{p! \times (2n-p)!}\right)$$

avec $p = 2$

2 Formule satisfaisable

2.1 Exemples

2.1.1 Formule satisfaisable

Une formule est dite satisfaisable si la formule est valué à Vrai.
Exemple (exemple du sujet) :

$$F = (x_1 \vee \neg x_2) \wedge (x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_4 \vee \neg x_5) \wedge (x_2 \vee \neg x_5)$$

Avec :
x_1 = True
x_2 = False
x_3 = True
x_4 = None
x_5 = False

2.1.2 Formule non satisfaisable (Exercice 2)

Inverssement une formule non satisfaisable est une formule valué à Faux
Exemple :

$$F = (x_4 \vee \neg x_3) \wedge (x_4 \vee x_1) \wedge (\neg x_2 \vee \neg x_1) \wedge (x_2 \vee \neg x_5) \wedge (\neg x_2 \vee x_5)$$

Avec : x_1 = True
x_2 = False
x_3 = True
x_4 = False
x_5 = False

2.2 Algorithme (Exercice 3)

Pour implémenter l'algorithme de satisfaisabilité nous avons décider de représenter l'algorithme avec une sémenthique particulière.
Tous d'abord la formule est entré dans une chaine de caractère.
Les clauses sont séparées par le symbole "&", et les variables à l'intérieur de la clause sont séparées par le symbole "|".
Lorsque la variable est précéder par "-" celà veut dire qu'il s'agit de la négation.
$1 : x_1$ -> $-1 : \neg x_1$
Ansi notre code fonctionne pour les formules 2-SAT. Par exemple en prenant la formule 2-SAT du sujet :
$F = (x_1 \vee \neg x_2) \wedge (x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_4 \vee \neg x_5) \wedge (x_2 \vee \neg x_5)$
On obtient

F="1|−2&3|4&−2|−3&4|−5&2|−5"

Les valeurs d'une varaibles sont affectée à l'aide d'un dictionnaire.
Chaque variable va pointer sur un booléen comme ceci :

```
dic={}
dic[1] = True
dic[2] = False
dic[3] = True
dic[4] = None
dic[5] = False
dic[6] = False
dic[-1] = not(dic[1])
dic[-2] = not(dic[2])
dic[-3] = not(dic[3])
dic[-4] = not(dic[4])
dic[-5] = not(dic[5])
dic[-6] = not(dic[6])
```

NB : Nous aurions également pu implémentez une fonction pour attribuer les négations mais ce n'étais pas une de nos priorités.

La chaine de caractère va être ensuite parsée.
Voici une ébauche de notre code permetant d'évaluer une formule

```
def clauses_liste(F)

def variables_par_clause(l1,l2)

def eval_clauses(d1,d2,l)

def eval_2_sat(f,d1):
    print(f"Affectation : {dic}")
    clauses = clauses_liste(f)
    print(f"Clauses : {clauses}")
    result = True
    clause=[]
    variables_par_clause(clause,clauses)
    print(f"Variables par clause : {clause}")
    d2={}
    eval_clauses(d1,d2,clause)
    print(f"Evaluations par clause : {d2}")
    for value in d2:
        result=(result and d2[value])
    return result
```

La fonction eval_2_sat prend en paramétre une formule f et un dictionnaire associé. Elle renvoit l'évaluation de la formule (en plus de quelques affichages pour faliciter la lecture).
Tous d'abord nous allons appeler la fonction clauses_liste qui prenant une formule 2-sat va renvoyer une liste (tableau) contenant les clauses.
L'impémentation ce fait en une ligne en effet nous utilisons seulement la fonction split comme ceci :

```
def clauses_liste(F):
    return F.split("&")
```

Nous splitons donc le tableau par le caractère "&".
Cette liste de clauses va être stocké dans une variable portant le même nom.
Ensuite nous appelons la procédure variables par clauses qui va ajouter dans une autre liste la liste des variables pas couples (même procédé que pour la fonction précédente).
Cette liste est appelé clause.

Ensuite nous allons créer un autre dictionnaire qui va affectée une évaluation pour chaque clause à l'aide de la fonction eval_clauses.

```
def eval_clauses(d1,d2,l):
    for i in range(len(l)):
        v1 = int(l[i][0])
        v2 = int(l[i][1])
        d2[tuple(l[i])] = (d1[v1] or d1[v2])
```

La fonction prend un argument une liste et son dictionnaire et un dictionnaire (vide en théorie).
Nous allons ensuites pour chaque clauses effectuer l'opération logique "ou" entre les deux variables.
On affecte ensuite le couple à un booléen (la valuation).
Enfin nous allons parcourir le dictionnaire qui affecte un booléen par couple (ici d2) pour effectuer l'opération logique "et" entre les différentes évaluations des couples.
Cette opération nous donne l'évaluation de la formule complète de la formule stockée dans result.

3 Graphe orienté associé à une formule

4 Algorithme pour vérifié si la formule est valide ou pas