3ieme année Génie INFORMATIQUE

Algorithme de résolution par réfutation selon la stratégie de résolution linéaire : implémentation en JAVA

Exposants:
HARA ZANG Ulrich
DJONGANG DJOMI Darylle

Sous la supervision de
MBOUH GHOGOMU Pryde

Table des matières

I- Algorithme 2

II- Justifications des choix d'implémentation 12

1- Structure de données :.....12

2- Classes et méthodes :.....12

III- Présentation des résultats 13

1- Test : 13

2- Indication pour test :.....28

IV- Conclusion 28

V- Bibliographie.....29

I- Algorithme

Nous présentons dans la suite un ensemble de fonctions nécessaire à la réalisation de l'algorithme de réfutation linéaire.

```
Booléen:Fonction compLit(Litteral l1, Litteral l2)
/*Objectif: fonction permettant de vérifier si deux littéraux sont complémentaires*/
/*Arguments: Deux enregistrements de type Litteral*/
/*Résultat: true si les deux littéraux passés en argument sont complémentaires et false sinon*/

//déclaration des variables

Variable:
        Booléen: b

Debut

        b←false
        Si(l1.l[0]='!') Alors
                Si(l2.l[0]='!') Alors
                        b←false
                Sinon
                        Si(l1.l[1]=l2.l[0]) Alors
                                b←true
                        Sinon
                                b←false
                        FinSi
                FinSi
        Sinon
                Si(l2.l[0]='!') Alors
                        Si(l1.l[0]=l2.l[1]) Alors
                                b←true
                        Sinon
                                b←false
                        FinSi
                Sinon
                        b←false
                FinSi
        FinSi

        renvoyer b
Fin
```

```
Booléen:Fonction equal(Litteral l1, Litteral l2)
/*Objectif: fonction permettant de vérifier si deux littéraux sont égaux*/
/*Arguments: Deux enregistrements de type Litteral*/
/*Résultat: true si les deux littéraux passés en argument sont égaux et false sinon*/

//déclaration des variables
Variable:
        Booléen: b

Debut

        Si(l1.l[0]=l2.l[0]) Alors
                Si(l1.l[1]=l2.l[1]) Alors
                        b←true
                Sinon
                        b←false
                FinSi
        Sinon
                b←false
        FinSi
        renvoyer b
Fin



Fonction duplicateLitteralsAvoidance(↓Clause clause)
/*Objectif: fonction permettant de supprimer dans une clause les doublons de littéraux*/
/*Arguments: un pointeur vers un enregistrement de type Clause*/

//déclaration des variables
Variable:
        Entier:i,j
        ArrayListLitteral: litteralsCopy

Debut

        i←0
        litteralsCopy=ArrayListLitteral(clause→litterals)
        Pour i←0 (1) size(clause→litterals)-2 Faire
                Pour j←i+1 (1) size(clause→litterals)-1 Faire
                        Si ( equal(get(clause→litterals,i), get(clause→litterals,j))) Alors
                                remove(litteralsCopy,i)
                        FinSi
                FinPour
        FinPour

        (clause→litterals)←ArrayListLitteral(litteralsCopy)

Fin
```

```
Booléen:Fonction paireReductible(Clause cl1,Clause cl2)
/*Objectif: fonction permettant de vérifier si la clause cl1 et la cl2 en paramètre sont une paire réductible*/
/*Arguments: deux enregistrements de type Clause*/
/*Résultat: true si les deux clauses sont une paire réductible et false sinon */

//déclaration des variables
Variables:
        Booléen:b
        Entier:nbLitCompl,i,j
        Litteral: l1,l2

Debut

        b←false
        nbLitCompl←0
        Pour i←0 (1) size(cl1.litterals)-1 Faire
                l1←get(cl1.litterals,i)
                Pour j←0 (1) size(cl2.litterals)-1 Faire
                        l2←get(cl2.litterals,j)
                        Si (compLit(l1,l2)) Alors
                                nbLitCompl←nbLitCompl+1
                        Finsi
                FinPour
        FinPour

        Si(nbLitCompl=0) Alors
                b←false
        FinSi
        Si(nbLitCompl=1) Alors
                b←true
        FinSi
        Si(nbLitCompl>=2) Alors
                b←false
        FinSi
        renvoyer b

Fin
```

```
↓Clause:Fonction resolvant(Clause cl1,Clause cl2)
/*Objectif: fonction permettant de trouver le résolvant de deux clauses ( qui sont une paire réductible)*/
/*Arguments: deux enregistrements de type Clause*/
/*Résultat: la clause résolvante, un pointeur vers un enregistrement de type Clause */

//déclaration des variables
Variables:

        Entier:i,j,k,l
        Litteral:l1,l2
        ArrayListLitteral:lit1,lit2


Debut

        i←0
        j←0
        lit1←ArrayListLitteral(cl1.litterals)
        lit2←ArrayListLitteral(cl2.litterals)

        Si(paireReductible(cl1,cl2)) Alors
                Pour k←0 (1) size(cl1.litterals)-1 Faire
                        l1←get(cl1.litterals,k)
                        Pour l←0 (1) size(cl2.litterals)-1 Faire
                                l2←get(cl2.litterals,l)
                                Si (compLit(l1,l2)) Alors
                                        i←indexOf(cl1.litterals,l1)
                                        j←indexOf(cl2.litterals,l2)
                                Finsi
                        FinPour
                FinPour
        remove(lit1,i)
        remove(lit2,j)
        addAll(lit1,lit2)
        renvoyer Clause(lit1)
        Sinon
                renvoyer NIL
        FinSi

Fin
```

```
Fonction addSommet(↓Graphe graphe, Clause clause)
/*Objectif: fonction d'ajout d'un sommet au graphe*/
/*Argument: un enregistrement de type Clause*/

//déclaration des variables
Variables:

Debut
        putIfAbsent(graphe, Sommet(clause), ArrayListSommet())
        Ecrire("Ajout du sommet: ",clause)
Fin



Fonction addArc(↓Graphe graphe, ↓Sommet s1, ↓Sommet s2)
/*Objectif: fonction permettant d'ajouter une arrête reliant les sommets s1 et s2*/
/*Argument: deux pointeurs vers des enregistrements de type Sommet et un pointeur vers un
enregistrement de type Graphe*/

//déclaration des variables
Variables:

Debut

        Si (s1!=NIL ET s2!=NIL) Alors
                Si(NON containsKey(graphe→adjs,s1) ) Alors
                        addSommet(s1)
                FinSi
                Si(NON containsKey(graphe→adjs,s2)) Alors
                        addSommet(s2)
                FinSi

                add(get(graphe→adjs,s1),s2)
                add(get(graphe→adjs,s2),s1)
                Ecrire("ajout d'arc entre ",getCl(s1))
                Ecrire(" et ",getCl(s2))
        Sinon
                Ecrire("Ajout d'arc impossible")
        FinSi

Fin
```

```
Booléen:Fonction verifDoublons(↓Graphe graphe, Clause clause)
/*Objectif: méthode permettant de vérifier si la clause passée en paramètre fait déjà parti des sommets du graphe.*/
/*Argument: un enregistrement de type Clause et un pointeur vers un enregistrement de type Graphe*/
/*Résultat: true si la clause est déjà existante dans le graphe et false sinon  */

//déclaration des variables
Variables:

        Booléen:b
        ArrayListSommet:sommets
        Sommet:sommet
        Entier:i

Debut

        b←false
        sommets←ArrayListSommet(keySet(graphe→adjs))

        Pour i←0 (1) size(sommets)-1 Faire
                sommet←get(sommets,i)
                Si (equal(getCl(sommet),clause)) Alors
                        return true
                FinSi
        FinPour
        b←false
        return b

Fin
```

```
Booléen:Fonction resolutionLineaire( ↓Graphe: graphe )
 /*Objectif: Réaliser l'algorithme de la résolution par réfutation avec la stratégie de résolution
linéaire*/
 /*Argument: un pointeur vers Enregistrement de type Graphe*/
 /*Résultat: retourne true s'il y a présence d'une clause vide et false sinon   */

 //déclaration des variables locales

Variable:

 Entier: i,j
 ↓ArrayListSommet: iterator
 ↓Sommet: c0,c1
 ↓Clause: resolvant


Debut

        iterator←keySet(getAdjs(graphe))
        c0←get(iterator, size(iterator)-1 )
        Ecrire("La clause centrale est c0: ",getCl(c0))
        resolvant←NIL

        Pour i←0 (1) size(iterator)-1 Faire
                j←i+1
                c1←get(iterator,i)
                Ecrire("c0: ",getCl(c0))
                Ecrire(" , c",j)
                Ecrire(": ",getCl(c1))
                resolvant←resolvant(getCl(c0), getCl(c1))

                Si (resolvant=NIL) Alors
                        Ecrire("paire non réductible")
                Sinon
                        Ecrire("resolvant : ", resolvant)
                FinSi

                Si(resolvant ≠ NIL) Alors

                        Si(isEmpty(getLitterals(resolvant))) Alors
                                b←true
                                renvoyer b
                        Sinon
                                Si (NON verifDoublons(resolvant)) Alors
                                        ↓Sommet r←Sommet(resolvant)
                                        Si (containsKey(getAdjs(graphe), r)) Alors
                                                addArc(graphe, c0, r)
                                                addArc(graphe, c1, r)
                                                resolutionLineaire(graphe)
                                                Si(b) Alors
                                                        break
```

```
                                                    Sinon
                                                        Ecrire("La clause n'ayant pas aboutit,
continuons avec la clause précédente: ",getCl(c0))
                                                    FinSi
                                                FinSi
                                            Sinon
                                                Ecrire("Cette clause existe déjà")
                                                Ecrire("On continue avec la clause centrale c0: ", getCl(c0))
                                            FinSi
                                    FinSi
                            Sinon
                                    Si(paireReductible(getCl(c0), getCl(c1)) Alors
                                            b←true
                                            renvoyer b
                                    Sinon
                                            Si(i=size(iterator)-1) Alors
                                                    b←false
                                                    renvoyer b
                                            FinSi
                                    Finsi
                            FinSi
                    FinPour
renvoyer b
Fin
```

```
Fonction enregistrerClauses(↓ArrayListClause clauses, ↓Graphe graphe)
/*Objectif: ajoute les clauses comme des sommets du graphe*/
/*Arguments: clauses un pointeur vers un enregistrement de type ArrayListClause, graphe un
pointeur vers un enregistrement de type graphe*/

//déclaration des variables

Variable:
        Clause: cl
        Entier: i

Debut

        Pour i←0 (1) clauses.size()-1 Faire
                cl←get(clauses,i)
                addSommet(cl)
        FinPour

Fin
```

```
Algorithme StrategieLineaire
/*Objectif: réaliser la résolution par réfutation selon la stratégie de résolution linéaire*/
/*Données: les clauses de la KB et ceux de la négation de la requête*/
/*Résultats: afficher true si KB^¬α est inconsistant et false s'il est consistant*/

/*Partie déclaration*/
Constante:
        MAX=2

Type:
        tLitt= Tableau[MAX + 1] de caractères

        Litteral=Enregistrement
                tLitt: l
        finEnregistrement

        Clause=Enregistrement
                ArrayListLitteral: litterals
        finEnregistrement

        Graphe=Enregistrement
                LinkedHashMapSommet: adjs
        finEnregistrement

        Sommet=Enregistrement
                Clause: cl
        finEnregistrement


Variable:
        Entier: nbClauses,i,j
        ↓ArrayListString: strings
        String: string
        ArrayListClauses: clauses
        ArrayListLitteral: litterals
        Litteral: litteral
        ↓Graphe: graphe
        Booléen : b
```

```
/*Partie exécution*/
Debut
        Ecrire("Entrer le nombre de clauses dans KB: ")
        Lire(nbClauses)
        Pour i←0 (1) nbClauses-1 Faire
                Ecrire("Entrer la clause c",i)
                Ecrire(": ")
                Lire(string)
                add(strings,string)
        FinPour

        Ecrire("Entrer le nombre de clauses dans !alpha: ")
        Lire(nbClauses)

        Pour i←0 (1) nbClauses-1
                Ecrire("Entrer la clause c",i)
                Ecrire(": ")
                Lire(string)
                add(strings,string)
        FinPour

        Pour j←0 (1) size(strings)-1 Faire
                string←get(strings,i)
                clear(litterals)
                tab←tocharArray(string)
                i←0

                Tantque (i<length(tab)) Faire
                        Si (©(tab+i) = '!') Alors
                                litteral←Litteral('!',©(tab+i+1))
                                add(litterals,litteral)
                                i←i+2
                        Sinon
                                Si (©(tab+i)≠' ') Alors
                                        litteral←Litteral(©(tab+i))
                                        add(litterals,litteral)
                                        i←i+1
                                Sinon
                                        i←i+1
                                FinSi
                        FinSi
                FinTantQue
                add(clauses,Clause(litterals))
        FinPour

        enregistrerClause(clauses,graphe)
        Ecrire(resolutionLineaire(graphe))

Fin
```

II- Justifications des choix d'implémentation

1- Structure de données :

Pour mettre en œuvre l'algorithme de résolution par réfutation selon la stratégie de résolution linéaire, nous avons choisi comme structure de données, les graphes. Et avons choisi de les conserver de façon dynamique grâce aux listes d'adjacence, car cette façon de faire offre la possibilité d'ajouter et de retirer des sommets et des arêtes. Quant à leur implémentation en Java nous avons opté pour des LinkedHashMap. Cette classe utilise une liste doublement chaînée contenant toutes les entrées de la table hachée, dans l'ordre où les clés ont été insérées dans la table, ce qui nous arrange dans nos choix dans l'algorithme. De plus, cette classe réalise la plupart des opérations en un temps constant (get(), containsKey(), next()).

2- Classes et méthodes :

Dans notre rapport, pour rendre l'implémentation plus aisée, nous avons opté pour une matérialisation des concepts sous-jacents à notre cadre, à savoir les concepts de clauses et de littéraux et avons de ce fait créé des classes, Clause et Litteral, référant à ces concepts. Ensuite pour les concepts liés au graphe et à sa manipulation on a créé les classes Graphe et Sommet. Et enfin pour la stratégie de résolution linéaire, on retrouve la classe StratégieLinéaire où se trouve la méthode principale resolutionLineaire() chargée de mettre en œuvre la stratégie par réfutation linéaire.

Pour une visualisation globale et profonde des choses nous avons décidé de présenter l'ensemble de notre package de travail grâce au Javadoc et un PDFDoclet. Donc vous trouverez joint un fichier qui présente ce dernier JavaDoc_sf_ia.

III- Présentation des résultats

1- Test :

a- Test1: KB={¬P v ¬Q , ¬Q v P } et ¬α= {Q, ¬P}

```
Entrer le nombre de clauses dans KB: 2
Entrer la clause c0: !P !Q
Entrer la clause c1: !Q P
Entrer le nombre de clauses dans !alpha:
2
Entrer la clause c0: Q
Entrer la clause c1: !P
Ajout du sommet :!P | !Q
Ajout du sommet :!Q | P
Ajout du sommet :Q
Ajout du sommet :!P

La clause centrale est c0: !P
c0: !P , c1: !P | !Q
paire non réductible
c0: !P , c2: !Q | P
resolvant : !Q

Ajout du sommet :!Q
ajout d'arc entre !P et !Q
ajout d'arc entre !Q | P et !Q

La clause centrale est c0: !Q
c0: !Q , c1: !P | !Q
paire non réductible
c0: !Q , c2: !Q | P
paire non réductible
c0: !Q , c3: Q
resolvant : La clause vide

true
```

On obtient true, donc on conclut que KB^ ⅂α est inconsistant.

b- Test2 : KB={⅂P ∨ ⅂Q ∨ R , ⅂R , P, ⅂T ∨ Q} et ⅂α=T

```
Entrer le nombre de clauses dans KB: 4
Entrer la clause c0: !P !Q R
Entrer la clause c1: !R
Entrer la clause c2: P
Entrer la clause c3: !T Q
Entrer le nombre de clauses dans !alpha:
1
Entrer la clause c0: T

Ajout du sommet :!P | !Q | R
Ajout du sommet :!R
Ajout du sommet :P
Ajout du sommet :!T | Q
Ajout du sommet :T

La clause centrale est c0: T

c0: T , c1: !P | !Q | R
paire non réductible
c0: T , c2: !R
paire non réductible
c0: T , c3: P
paire non réductible
c0: T , c4: !T | Q
resolvant : Q

Ajout du sommet :Q
ajout d'arc entre T et Q
ajout d'arc entre !T | Q et Q
```

```
La clause centrale est c0: Q

c0: Q , c1: !P | !Q | R
resolvant : !P | R

Ajout du sommet :!P | R
ajout d'arc entre Q et !P | R
ajout d'arc entre !P | !Q | R et !P | R

La clause centrale est c0: !P | R

c0: !P | R , c1: !P | !Q | R
paire non réductible
c0: !P | R , c2: !R
resolvant : !P

Ajout du sommet :!P
ajout d'arc entre !P | R et !P
ajout d'arc entre !R et !P

La clause centrale est c0: !P

c0: !P , c1: !P | !Q | R
paire non réductible
c0: !P , c2: !R
paire non réductible
c0: !P , c3: P
resolvant : La clause vide

true
```

On obtient true, donc on conclut que KB^ ٦α est inconsistant.

c- Test3: KB={¬A , ¬B ∨ C ∨ D, ¬C ∨ B, ¬D ∨ B, ¬E ∨ A ∨ F ∨ G, ¬A ∨ E , ¬F ∨ E, ¬G ∨ E, ¬B , E} et ¬α=C

```
Entrer le nombre de clauses dans KB: 10
Entrer la clause c0: !A
Entrer la clause c1: !B C D
Entrer la clause c2: !C B
Entrer la clause c3: !D B
Entrer la clause c4: !E A F G
Entrer la clause c5: !A E
Entrer la clause c6: !F E
Entrer la clause c7: !G E
Entrer la clause c8: !B
Entrer la clause c9: E
Entrer le nombre de clauses dans !alpha:
1
Entrer la clause c0: C
Ajout du sommet :!A
Ajout du sommet :!B | C | D
Ajout du sommet :!C | B
Ajout du sommet :!D | B
Ajout du sommet :!E | A | F | G
Ajout du sommet :!A | E
Ajout du sommet :!F | E
Ajout du sommet :!G | E
Ajout du sommet :!B
Ajout du sommet :E
Ajout du sommet :C

La clause centrale est c0: C

c0: C , c1: !A
paire non réductible
c0: C , c2: !B | C | D
```

c0: C , c3: !C | B

resolvant : B

Ajout du sommet :B

ajout d'arc entre C et B

ajout d'arc entre !C | B et B

La clause centrale est c0: B

c0: B , c1: !A

paire non réductible

c0: B , c2: !B | C | D

resolvant : C | D

Ajout du sommet :C | D

ajout d'arc entre B et C | D

ajout d'arc entre !B | C | D et C | D

La clause centrale est c0: C | D

c0: C | D , c1: !A

paire non réductible

c0: C | D , c2: !B | C | D

paire non réductible

c0: C | D , c3: !C | B

resolvant : D | B

Ajout du sommet :D | B

ajout d'arc entre C | D et D | B

ajout d'arc entre !C | B et D | B

La clause centrale est c0: D | B

c0: D | B , c1: !A

paire non réductible

c0: D | B , c2: !B | C | D

resolvant : C | D

Cette clause existe déjà

On continue avec la clause centrale c0: D | B

c0: D | B , c3: !C | B

paire non réductible

c0: D | B , c4: !D | B

resolvant : B

Cette clause existe déjà

On continue avec la clause centrale c0: D | B

c0: D | B , c5: !E | A | F | G

paire non réductible

c0: D | B , c6: !A | E

paire non réductible

c0: D | B , c7: !F | E

paire non réductible

c0: D | B , c8: !G | E

paire non réductible

c0: D | B , c9: !B

resolvant : D

Ajout du sommet :D

ajout d'arc entre D | B et D

ajout d'arc entre !B et D

La clause centrale est c0: D

c0: D , c1: !A

paire non réductible

c0: D , c2: !B | C | D

paire non réductible

c0: D , c3: !C | B

paire non réductible

c0: D , c4: !D | B

resolvant : B

Cette clause existe déjà

On continue avec la clause centrale c0: D

c0: D , c5: !E | A | F | G

paire non réductible

c0: D , c6: !A | E

paire non réductible

c0: D , c7: !F | E

paire non réductible

c0: D , c8: !G | E

paire non réductible

c0: D , c9: !B

paire non réductible

c0: D , c10: E

paire non réductible

c0: D , c11: C

paire non réductible

c0: D , c12: B

paire non réductible

c0: D , c13: C | D

paire non réductible

c0: D , c14: D | B

paire non réductible

La clause n'ayant pas aboutit, continuons avec la clause précédente: D | B

c0: D | B , c10: E

paire non réductible

c0: D | B , c11: C

paire non réductible

c0: D | B , c12: B

paire non réductible

c0: D | B , c13: C | D

paire non réductible

La clause n'ayant pas aboutit, continuons avec la clause précédente: C | D

c0: C | D , c4: !D | B

resolvant : C | B

Ajout du sommet :C | B

ajout d'arc entre C | D et C | B

ajout d'arc entre !D | B et C | B

La clause centrale est c0: C | B

c0: C | B , c1: !A

paire non réductible

c0: C | B , c2: !B | C | D

resolvant : C | D

Cette clause existe déjà

On continue avec la clause centrale c0: C | B

c0: C | B , c3: !C | B

resolvant : B

Cette clause existe déjà

On continue avec la clause centrale c0: C | B

c0: C | B , c4: !D | B

paire non réductible

c0: C | B , c5: !E | A | F | G

paire non réductible

c0: C | B , c6: !A | E

paire non réductible

c0: C | B , c7: !F | E

paire non réductible

c0: C | B , c8: !G | E

paire non réductible

c0: C | B , c9: !B

resolvant : C

Cette clause existe déjà

On continue avec la clause centrale c0: C | B

c0: C | B , c10: E

paire non réductible

c0: C | B , c11: C

paire non réductible

c0: C | B , c12: B

paire non réductible

c0: C | B , c13: C | D

paire non réductible

c0: C | B , c14: D | B

paire non réductible

c0: C | B , c15: D

```
paire non réductible

La clause n'ayant pas aboutit, continuons avec la clause précédente: C | D

c0: C | D , c5: !E | A | F | G
paire non réductible
c0: C | D , c6: !A | E
paire non réductible
c0: C | D , c7: !F | E
paire non réductible
c0: C | D , c8: !G | E
paire non réductible
c0: C | D , c9: !B
paire non réductible
c0: C | D , c10: E
paire non réductible
c0: C | D , c11: C
paire non réductible
c0: C | D , c12: B
paire non réductible

La clause n'ayant pas aboutit, continuons avec la clause précédente: B

c0: B , c3: !C | B
paire non réductible
c0: B , c4: !D | B
paire non réductible
c0: B , c5: !E | A | F | G
paire non réductible
c0: B , c6: !A | E
paire non réductible
c0: B , c7: !F | E
paire non réductible
c0: B , c8: !G | E
paire non réductible
c0: B , c9: !B
resolvant : La clause vide

true
```

On obtient true, donc on conclut que KB^ ٦α est inconsistant. Ce qui est important de remarquer dans ce test est que, lorsqu'une clause centrale n'aboutit pas on revient à la clause centrale précédente pour continuer là où nous sommes arrêtés pour essayer de trouver un autre résolvant et continuer ainsi avec lui.

d- Test4: KB={B ∨ ¬D ∨ ¬C, ¬B ∨ ¬C, ¬B ∨ D, M ∨ ¬C ∨ D , ¬M ∨ C ∨ B , C} et ¬α=¬B

```
Entrer le nombre de clauses dans KB: 6
Entrer la clause c0: B !D !C
Entrer la clause c1: !B !C
Entrer la clause c2: !B D
Entrer la clause c3: M !C D
Entrer la clause c4: !M C B
Entrer la clause c5: C
Entrer le nombre de clauses dans !alpha:
1
Entrer la clause c0: !B

Ajout du sommet :B | !D | !C
Ajout du sommet :!B | !C
Ajout du sommet :!B | D
Ajout du sommet :M | !C | D
Ajout du sommet :!M | C | B
Ajout du sommet :C
Ajout du sommet :!B

La clause centrale est c0: !B
c0: !B , c1: B | !D | !C
resolvant : !D | !C

Ajout du sommet :!D | !C
ajout d'arc entre !B et !D | !C
ajout d'arc entre B | !D | !C et !D | !C

La clause centrale est c0: !D | !C

c0: !D | !C , c1: B | !D | !C
paire non réductible
c0: !D | !C , c2: !B | !C
paire non réductible
c0: !D | !C , c3: !B | D
resolvant : !C | !B

Cette clause existe déjà
On continue avec la clause centrale c0: !D | !C

c0: !D | !C , c4: M | !C | D
resolvant : M | !C

Ajout du sommet :M | !C
ajout d'arc entre !D | !C et M | !C
ajout d'arc entre M | !C | D et M | !C
```

La clause centrale est c0: M | !C
c0: M | !C , c1: B | !D | !C
paire non réductible
c0: M | !C , c2: !B | !C
paire non réductible
c0: M | !C , c3: !B | D
paire non réductible
c0: M | !C , c4: M | !C | D
paire non réductible
c0: M | !C , c5: !M | C | B
paire non réductible
c0: M | !C , c6: C
resolvant : M

Ajout du sommet :M
ajout d'arc entre M | !C et M
ajout d'arc entre C et M

La clause centrale est c0: M
c0: M , c1: B | !D | !C
paire non réductible
c0: M , c2: !B | !C
paire non réductible
c0: M , c3: !B | D
paire non réductible
c0: M , c4: M | !C | D
paire non réductible
c0: M , c5: !M | C | B
resolvant : C | B
Ajout du sommet :C | B
ajout d'arc entre M et C | B
ajout d'arc entre !M | C | B et C | B

La clause centrale est c0: C | B
c0: C | B , c1: B | !D | !C
resolvant : B | !D

Ajout du sommet :B | !D
ajout d'arc entre C | B et B | !D
ajout d'arc entre B | !D | !C et B | !D
La clause centrale est c0: B | !D
c0: B | !D , c1: B | !D | !C
paire non réductible
c0: B | !D , c2: !B | !C
resolvant : !D | !C

Cette clause existe déjà
On continue avec la clause centrale c0: B | !D

c0: B | !D , c3: !B | D
paire non réductible
c0: B | !D , c4: M | !C | D
resolvant : B | M | !C

Ajout du sommet :B | M | !C
ajout d'arc entre B | !D et B | M | !C
ajout d'arc entre M | !C | D et B | M | !C

La clause centrale est c0: B | M | !C
c0: B | M | !C , c1: B | !D | !C
paire non réductible
c0: B | M | !C , c2: !B | !C
resolvant : M | !C

Cette clause existe déjà
On continue avec la clause centrale c0: B | M | !C
c0: B | M | !C , c3: !B | D
resolvant : M | !C | D

Cette clause existe déjà
On continue avec la clause centrale c0: B | M | !C
c0: B | M | !C , c4: M | !C | D
paire non réductible
c0: B | M | !C , c5: !M | C | B
paire non réductible
c0: B | M | !C , c6: C
resolvant : B | M

Ajout du sommet :B | M
ajout d'arc entre B | M | !C et B | M
ajout d'arc entre C et B | M

La clause centrale est c0: B | M
c0: B | M , c1: B | !D | !C
paire non réductible
c0: B | M , c2: !B | !C
resolvant : M | !C

Cette clause existe déjà
On continue avec la clause centrale c0: B | M
c0: B | M , c3: !B | D
resolvant : M | D
Ajout du sommet :M | D
ajout d'arc entre B | M et M | D
ajout d'arc entre !B | D et M | D

La clause centrale est c0: M | D

Pour des soucis de longueur nous n'avons pas afficher les étapes intermédiaires à ce niveau, mais le lecteur pourra saisir les données en entrées et vérifier ainsi ces étapes intermédiaires.

```
c0: !D | !M | B , c17: C | D
resolvant : !M | B | C

Cette clause existe déjà
On continue avec la clause centrale c0: !D | !M | B

c0: !D | !M | B , c18: D | B | M
paire non réductible
c0: !D | !M | B , c19: !D
paire non réductible

La clause n'ayant pas aboutit, continuons avec la clause précédente: !D | !C

c0: !D | !C , c6: C
resolvant : !D

Cette clause existe déjà
On continue avec la clause centrale c0: !D | !C

c0: !D | !C , c7: !B
paire non réductible

La clause n'ayant pas aboutit, continuons avec la clause précédente: !B

c0: !B , c2: !B | !C
paire non réductible
c0: !B , c3: !B | D
paire non réductible
c0: !B , c4: M | !C | D
paire non réductible
c0: !B , c5: !M | C | B
resolvant : !M | C

Cette clause existe déjà
On continue avec la clause centrale c0: !B

c0: !B , c6: C
paire non réductible
false
```

On obtient false, donc on conclut que KB^ ٦α est consistant. Ici il faut remarquer que l'on a testé vraiment toutes les possibilités entre les différentes clauses centrales et les autres clauses sans pouvoir aboutir.

e- Test5: KB={¬S, ¬P v Q, P} et ¬α=RvQ

```
Entrer le nombre de clauses dans KB: 3
Entrer la clause c0: !Q
Entrer la clause c1: !P Q
Entrer la clause c2: P
Entrer le nombre de clauses dans !alpha:
1
Entrer la clause c0: R Q


Ajout du sommet :!Q
Ajout du sommet :!P | Q
Ajout du sommet :P
Ajout du sommet :R | Q

La clause centrale est c0: R | Q

c0: R | Q , c1: !Q
resolvant : R

Ajout du sommet :R
ajout d'arc entre R | Q et R
ajout d'arc entre !Q et R

La clause centrale est c0: R

c0: R , c1: !Q
paire non réductible
c0: R , c2: !P | Q
paire non réductible
c0: R , c3: P
paire non réductible
c0: R , c4: R | Q
paire non réductible

La clause n'ayant pas aboutit, continuons avec la clause précédente: R | Q

c0: R | Q , c2: !P | Q
paire non réductible
c0: R | Q , c3: P
paire non réductible
false
```

On obtient false, donc on conclut que KB^ ¬α est consistant. Ici il faut aussi remarquer que l'on a testé vraiment toutes les possibilités entre les différentes clauses centrales et les autres clauses sans pouvoir aboutir.

2- Indication pour test :

- On demande à l'utilisateur d'entrer le nombre de clauses de la KB, il doit saisir un entier strictement positif
- Ensuite on lui demande d'entrer les clauses de cette KB au fur et à mesure. Les indications pour celles-ci sont :
    - Pour une clause les littéraux doivent être séparés par un espace exemple : P Q R (ce qui correspond à $P \vee Q \vee R$
    - La négation d'un littéral est : !
    - Pour valider la saisi d'une clause appuyer sur la touche « Enter »
    - Un exemple complet de saisi d'une clause est donc : !P Q !R S « Enter » (ce qui correspond à $\neg P \vee Q \vee \neg R \vee \neg S$
- Puis l'on demande d'entrer le nombre de clauses de la négation de la requête (!alpha), il doit une fois de plus saisir un entier strictement positif.
- Et enfin on lui demande d'entrer les clauses de la négation de la requête (voir les indications données au point 2 pour la saisi des clauses).

IV- Conclusion

Tout au long de notre travail nous nous sommes attardés sur l'algorithme de résolution par réfutation, selon la stratégie de résolution linéaire. Au terme de notre analyse il ressort que cet algorithme offre effectivement des résultats vraiment pertinents et est de loin plus efficace que les stratégies en « largeur » et de l' « ensemble-support » quand elle aboutit. Mais elle est incomplète, c'est-à-dire qu'il se peut qu'elle ne permette pas d'aboutir à la clause vide alors que l'ensemble de clauses est contradictoire (doit normalement aboutir). Ceci est dû, en général, à de mauvais choix dans l'utilisation des clauses. Pour limiter ce problème, on prend souvent comme clause centrale de départ une clause issue de la négation de la conclusion ; il reste à faire les bons choix pour les clauses de bord. Autrement dit le choix des clauses de bords joue aussi pour beaucoup pour que l'algorithme aboutisse. Toutefois il existe un moyen pour rendre la résolution linéaire (par entrée) complète, il suffit de prendre des clauses de Horn. Autrement dit la résolution linéaire (par entrée) est complète pour la réfutation des clauses de Horn.

V- Bibliographie

P. M. Ghogomu, « Chapitre 1 : la logique propositionnelle »

« Graphes en Java ». https://www.codeflow.site/fr/article/java-graphs (consulté le juin 15, 2020).

« ResProp ». https://pages.lip6.fr/Jean-Francois.Perrot/Prolog/Cours3/ResProp.html