GUERROUDJ David BERTOLINI Alban

Intelligence Artificielle

Projet - Master 1 Informatique

[](http://www.univ-amu.fr/)

# Table des matières

Voici le projet d’intelligence Artificielle de Mr GUERROUDJ et Mr BERTOLINI

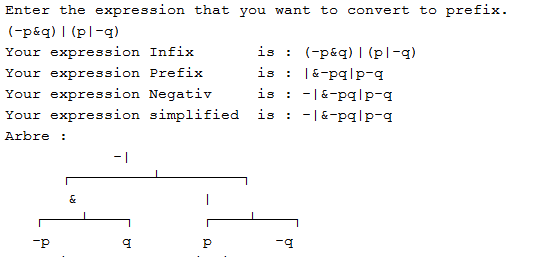
1. Table des matières
2. Présentation du Projet
3. Exemple d’utilisation
4. Explication de notre algorithme
5. Présentation des différentes classes
6. Problèmes rencontrés
7. Conclusion

# Présentation du Projet

Le but de notre projet est de créer un programme capable de **[A COMPLETER]**

# Exemple d’utilisation

Voici un exemple d’utilisation du programme :



Ici, on observe que le programme nous demande de rentrer une formule. Nous rentrons ici la formule : (-p&q)|(p|-q).

Durant la première partie nous voyons les différents traitements de la formule, en forme Infixe, préfixe, et négative. Vient ensuite toute les simplifications du négatif.  
Toutes les transformations effectuées, on observe finalement l’arbre formé.

**[A COMPLETER]**

# Explications de notre algorithme

Dans un premier temps nous prenons en paramètre de notre fonction une formule que nous stockons par la suite dans un String. Une fois ce string récupéré, nous le traitons de diverses manières afin d’obtenir une forme préfixe, afin de créer un arbre, pratique et facile pour nous à explorer. Cet arbre nous est indispensable afin de bien parcourir notre formule et d’y appliquer nos règles. Une fois cet arbre créé, il nous suffit d’y appliquer notre algorithme, qui va d’abord parcourir notre arbre et y appliquer les différentes règles.

**[A COMPLETER]**

# Présentation des différentes classes

Dans cette partie nous vous décrirons les grandes lignes de toutes les classes que nous décidons d’utiliser dans notre programme.

* Prefixer.java

La classe Prefixer.java est une classe qui va nous permettre de pouvoir traiter n’importe quelle formule de la forme (L**x**L)**x**(L**x**L) ou bien L**x**L où L ∈ { A,B,C…Z } \ { L,M } c’est-à-dire un littéral précédé d’un symbole unaire { -,L,M } où L et M sont les symboles (resp.□ et ◊) et où x est un symbole { & , | , > } (resp. et, ou, implique).  
A partir de la chaine de caractère dans laquelle est stockée la formule, nous allons commencer par créer un Arbre Binaire, dans lequel nous plaçons les différents littéraux et les différents symboles dans des nœuds afin de la passer d’une forme infixe, à une forme préfixe.

Par exemple nous avons la formule A & B :

&zze

Azze

Bzze

Notre algorithme va nous permettre de transformer ensuite cet Arbre de cette façon :

Azze

&zze

Bzze

Pour finalement donner une formule sous la forme préfixe : &AB

Nous avons aussi la possibilité de rendre cette formule négative.

Notre algorithme nous permet de traduire n’importe quelle formule sous cette forme à partir du moment qu’elle est sous forme binaire. Il faut donc utiliser des parenthèses pour éviter tout bug.  
Nous nous retrouvons donc au final avec 3 variables contenant respectivement la forme infixe, la forme préfixe et la forme négative :

1. infixFormula
2. prefixFormula
3. negativeFormula

* Formula.java

La classe formula est une classe qui va permettre de pouvoir directement traiter une chaine de caractère dans laquelle nous plaçons une formule. Elle est composée de fonctions qui permettent de traiter ces chaines de caractères en leur appliquant des formules de logique propositionnelle.  
Nous avons ici la possibilité dans un premier temps de pouvoir simplifier les « - » devant les littéraux et les symboles unaire.

Par exemple : --A donnera A.

Nous sommes aussi en mesure de pouvoir factoriser les symboles unaires présent une formule.

Par exemple LA&LB donnera L&AB.

Et enfin, nous avons la possibilité grâce à cette classe de pouvoir extraire tous les membres d’une formule qui peuvent, après toute simplification, être placé dans un nœud de notre arbre final.

Cette classe est nécessaire pour la suite de notre programme, elle va permettre de nous faciliter la création de notre arbre.

* Node.java

La classe Node est une simple classe qui contient une structure qui va nous permettre de construire un Arbre dans lequel nous pouvons stocker une formule propositionnelle de la forme infixe. Il s’agit ici d’un arbre Binaire dans lequel chaque nœud possède au maximum deux fils.

* BinaryTree.java

La classe BinaryTree est une classe qui va nous permettre de construire un Arbre Binaire de la même forme que le premier Arbre que nous utilisons dans la classe prefixer. Mais qui contrairement à la classe précédente, va se servir de la classe Formula afin de créer l’arbre le plus optimal à nos besoins.  
  
Par exemple la formule : |&LALB&CD

Nous donnera un arbre de la forme suivante à l’aide de la classe prefixer.

|zze

&zze

&zze

LB

F

C

LAAzze

Mais notre nouvelle classe contient des fonctions capables de créer un arbre qui possèdent des nœuds qui contiennent en plus d’un littéral, une suite de symboles unaire factorisés.

Nous obtiendrons donc un arbre de la forme suivante :

|zze

L&zze

&zze

F

C

B

Azze

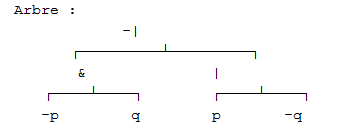
Ici, nous observons bien que le L à été factorisé au nœud précédent.

Le but de cet arbre est de pouvoir parcourir notre formule le plus simplement possible et ainsi de directement y appliquer les différentes formules dont nous aurons besoin par la suite.

* TreePrinters.java

La classe TreePrinters est une simple classe qui permet d’afficher graphiquement dans une console un arbre binaire.   
  
Prenons par exemple la formule : (-p&q)|(p|-q)

Cette classe va permettre de l’afficher de la forme suivante :



* Branch.java

La classe Branch est une classe qui contient la structure que nous utilisons dans notre programme afin de résoudre le problème donnée.

* NodeCB.java

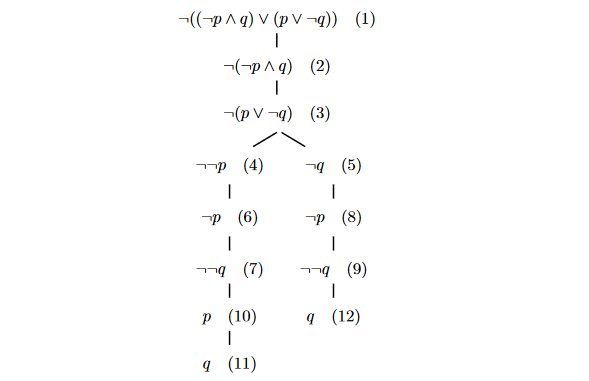
La classe NodeCB est une classe qui contient la structure que nous utilisons dans notre programme afin de résoudre

* Rules.java

La classe Rules est une classe qui contient toutes les règles que nous utiliserons par la suite dans notre algorithme. Telles que les règles :

* + - * 1. AND
        2. NOT AND
        3. OR
        4. NOT OR
        5. IMPLIQUE
        6. NOT IMPLIQUE
        7. L
        8. NOT L
* ClosingBranch.java

La classe ClosingBranch est la classe qui contient toutes les fonctions liées à l’algorithme que nous utilisons. Il contient des fonctions qui permettent de détecter qu’une branche contient des contradictions, de créer notre dernière structure de donnée :



* Main.java

La classe Main est la classe qui va permettre l’exécution de tout notre programme. Il contient tout simplement le main de notre programme.

# Problèmes rencontrés

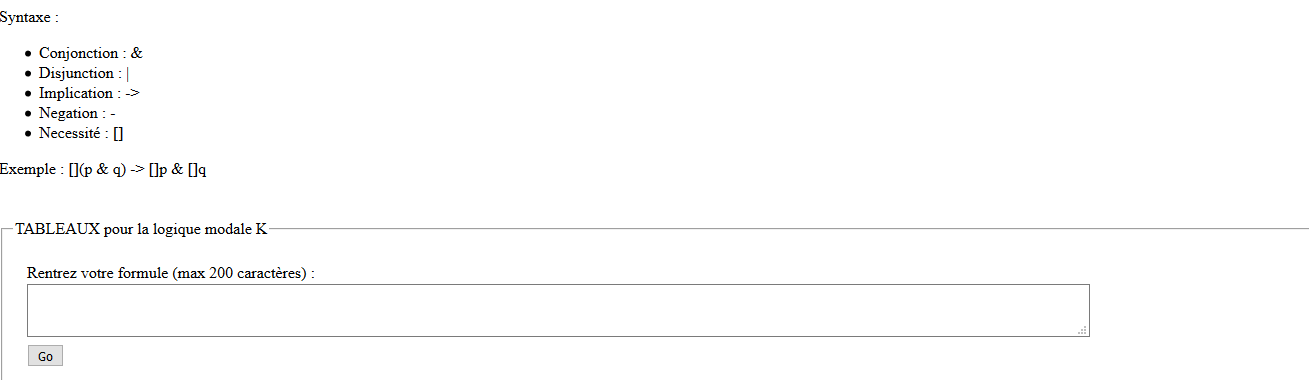
Durant toute l’élaboration de notre programme, nous nous sommes retrouvés confronté à divers problèmes  
Le choix du langage a tout d’abord été un dilemme conséquent. Nous avons commencé par coder la fonction de conversion infixe 2 préfix en langage C, mais face à des soucis concernant cette fonction nous avons décidé de passer en langage JAVA.

Le premier vrai problème auquel nous avons fait face, a été de créer de comprendre avec exactitude et découvrir quelles seraient les structures les mieux adaptées à la consigne.

Ensuite vint le problème de l’arbre. Nous avions commencé par créer un arbre simple comportant la formule, mais il était alors difficile de pouvoir utiliser la formule convenablement afin d’y utiliser les règles demandées dans le sujet. C’est pourquoi nous avons décidé de commencer par la création de formule de formatage de la formule (cf. factorisation, simplification...).

Cela nous a permis de finalement avoir une structure propre et adaptée à nos besoins.

Une fois le problème du langage et des structures résolus, il ne nous restait qu’a bien comprendre comment l’algorithme fonctionnait et de l’implémenter. Pour cela nous nous sommes servis du programme de Mr SANTOCANALE et des différentes planches de TD afin de l’appliquer convenablement.



# Conclusion

**A COMPLETER**