UNIVERSITÉ KASDI MERBAH OUARGLA

Faculté des Nouvelles Technologies de l'Information et de la Communication

Département d'Informatique et des Technologies de l'Information

Mémoire Master en Informatique

Domaine : Mathématique et Informatique

Filière : Informatique

Spécialité : Administration et Sécurité des Réseaux

Présenté par : Ghoula Meriem et Chiba Raouia

Implémentation d'un Outil de Raisonnement Propositionnel basé sur La Méthode de Résolution par Réfutation

Soutenue le : 21/06/2021

Devant le jury composé de :

Dr Toumi Chahrazed	UKM Ouargla	Président du jury
Dr Chama Wafa	UKM Ouargla	Superviseur
Dr Marref Nadia	UKM Ouargla	Membre du jury

Année universitaire : 2020/2021

Remerciement

Tout d'abord, nous remercions «Allah », le très-haut, le grand, le compatissant, le miséricordieux, qui nous a donne la santé pour faire cette humble œuvre.

Nous remercions tout particuliers notre superviseur Chama Wafa qui nous a fournis des informations précieuses et nous a aidés étape par étape à atteindre la fin.

Nous exprimons notre remerciement à tous les enseignants du département d'informatique et des sciences d'information de l'université d'Ouargla sur la connaissance que nous avons reçues.

Nous remercions également les membres du jury d'avoir participé à la discussion de la fin de notre étude, et enfin nous remercions tous ceux qui ont contribué directement ou indirectement à la réalisation de ce travail.

Dédicaces

Je dédie ce travail avec mon respect et ma gratitude

À la source de l'amour et de la tendresse …… à mon paradis ….. Ma mère.

Aux ceux qui m'ont inculqué l'ambition et la persévérance …….. Mon père.

A ceux qui achèvent ma joie ……. mes frères et sœurs.

Aux cœurs qui nous contiennent quand le monde nous rétrécit …….. ma famille.

Aux tous ceux qui m'ont aidé et qui se tenaient à mes cotés.

A tous ceux qui ont contribué à raviver l'espoir en moi-même et qui ont cultivé confiance et optimisme.

Aux l'enfance volée et l'innocence assassinée ……. les enfants de Palestine.

Aux amateurs de science et de connaissance.

Ghoula Meriem

Dédicaces

À mes chers parents, qui m'ont toujours donné le meilleur de l'amour et du sacrifice. À mes frères et sœurs, toute ma famille et mes amis, pour les soutenir tous tout au long de ma carrière universitaire. À toute personne proche ou lointaine donnez-moi du soutien et des encouragements pour atteindre mon objectif, tous ceux qui m'ont appris même une lettre d'enseignants et de professeurs.

Chiba Raouia

Résumé

La logique mathématique est un langage de représentation de l'information pour aboutir des résultats et des conclusions. C'est la science qui étudie les règles générales du raisonnement correct. La logique contient de nombreux types différents, le calcul propositionnel est la plus simple, il est utilisé dans tous les domaines de l'informatique. La logique occupe une grande place dans le développement de matériel informatique et la construction de ses applications, tous les langages de programmation utilisent les connecteurs logiques.

Ce projet de fin d'étude propose une approche idéale afin d'étudier les propriétés des formules logiques, où un outil a été développé basé sur l'une des méthodes de preuve, la méthode de résolution par réfutation. Notre objectif est de fournir des résultats précis, non ambigus, réduire l'effort et le temps requis par les tables de vérité.

Mots clés : logique propositionnelle, formule, méthodes de preuve, méthode de résolution, réfutation.

Abstract

Mathematical logic is a language for representing information in order to arrive at results and conclusions. It is the science that studies the general rules of correct reasoning. Logic contains many different types, propositional calculus is the simplest, and it is used in all areas of computer science. Logic occupies a large place in the development of computer hardware and the construction of its applications; all programming languages use logical connectors.

This end-of-study project offers an ideal approach to study the properties of logical formulas, where a tool has been developed based on one of the proof methods, the resolution method with refutation. Our goal is to provide accurate, unambiguous results, reducing the effort and time required by truth tables.

Keywords: propositional logic, formula, proof methods, resolution method, refutation.

ملخص

المنطق الرياضي هو لغة تمثيل للمعلومات من أجل التحصل على النتائج والاستنتاجات. هو العلم الذي يدرس قواعد التفكير الصحيح. المنطق يتكون من عدة أنواع من بينها المنطق الافتراضي و هو الأكثر بساطة. يستخدم المنطق في جميع مجالات الحوسبة حيث يحتل مكانة كبيرة في تطوير أجهزة الحاسوب و بناء تطبيقاته اذ كل لغات البرمجة تستخدم الروابط المنطقية.

مشروع نهاية الدراسة يقترح نهجا مثاليا لدراسة خصائص الصيغ المنطقية حيث تم تطوير أداة بناءا على إحدى طرق الإثبات ألا وهي طريقة الحل عن طريقة التفنيد. هدفنا هو تقديم نتائج دقيقة لا لبس فيها وتقليل الجهد والوقت الذي يتطلبهما جداول الحقيقة.

الكلمات المفتاحية : المنطق الافتراضي, صيغة, طرق الإثبات, طريقة الدقة, التفنيد .

Table de matière

Liste des tableaux

Liste des figures

Liste des abréviations

LP : logique propositionnelle.

CP : calcule propositionnelle.

FP : formule propositionnelle.

FBF : formule bien formée.

SF : sous-formule.

FNN : forme normale négative.

FNC : forme normale conjonctive.

FND : forme normale disjonctive.

Introduction Générale

La logique est la science de la pensée attentive, leur but essentiel est de parvenir à des conclusions précises et non ambiguës dans un domaine particulier. En d'autres termes, les logiciens se sont toujours intéressés aux propriétés formelles de l'inférence valide, cependant, elle n'a pas été considéré comme faisant partie des mathématiques.

A la fin du 19 ^{e} siècle, la logique est devenue une partie de l'étude des mathématiques et est même considérée comme une science. Cela, grâce au scientifique anglais *George Poole* qui a été en mesure de développer la logique pour devenir un système mathématique abstrait et l'a utilisé comme un type spécial d'algèbre plus tard appelé Algèbre de Boole. Ce dernier est devenu une partie essentielle de la conception des circuits logiques dans les ordinateurs et ses bases ont beaucoup contribué dans les applications de l'intelligence artificielle.

La science logique est indispensable lors de l'étude des branches de différentes sciences, elle montre comment vous pensez correctement et tirez des résultats des données et comment vous soutenez votre point de vue avec les preuves correctes.

Les logiques classiques (la logique des propositions et la logique des prédicats) ont été créées pour raisonner sur des objets mathématiques qui sont conceptuellement assez simples. Les systèmes déductifs de la logique classique se limitent à la formalisation du raisonnement valide, simplement appelée logique à ses début. L'émergence des autres systèmes logiques formels, en particulier pour la logique intuitive, a incité l'ajout de l'adjectif classique au terme logique.

Le calcul des propositions est la base de toute définition logique et même de tout raisonnement. C'est aussi la première étape de la construction du calcul des prédicats, qui lui s'intéresse au contenu des propositions. Le calcul des propositions est parfois appelé *logique des propositions*, *logique propositionnelle*, *calcul des énoncés*, *théorie des fonctions de vérité* et aussi *logique booléenne*. Le calcul des énoncés tient une grande place en informatique, parce que nos ordinateurs actuels sont digitaux, et travaillent en binaire. Ce qui fait que nos processeurs sont essentiellement constitués de portes binaires.

Le but du calcul propositionnel est de donner un fondement formel à un ensemble restreint d'énoncés du langage et d'étudier les relations entre eux. Les énoncés traitées sont appelées propositions ou encore des formules. Les relations peuvent être exprimées par l'intermédiaire de connecteurs logiques qui permettent de construire des formules syntaxiquement correctes.

La logique propositionnelle sert à étudier la valeur de vérité d'une formule pour toutes les valuations possibles des variables qu'elle contient, cela afin de prouver certaines propriétés. Bien qu'il soit possible d'énumérer toutes les évaluations pour créer une table de vérité, la méthode est coûteuse en espace et en temps, surtout lorsque le nombre de variables est important ou parfois inconnu. Afin de surmonter ce problème, il est préférable d'utiliser des méthodes de preuve qui aident à réduire le temps, l'espace et surtout elle sert de donner des résultats précis et non ambigus. Il existe de nombreuses méthodes de preuve, chacune se distingue par son propre principe, et parmi d'elles nous nous distinguons en mentionnant la méthode de résolution qui traite les formules par réfutation. Cette méthode permet de

démontrer la validité d’une formule en démontrant l’inconsistance de sa négation. Nous pouvons appliquer la méthode de résolution à n'importe quel ensemble de formules.

Sur la base de ce qui précède, nous avons organisé notre mémoire en trois chapitres. Le premier introduit les notions essentielles de la logique propositionnelle en expliquant ses aspects syntaxique et sémantique. Nous avons essayé de clarifier les caractéristiques de chaque côté en identifiant ses éléments les plus importants.

Le deuxième chapitre propose un ensemble de méthodes de preuve qui aident à déterminer la valeur de vérité des formules non simples, plus compliquées, qui demandent bien sûr beaucoup de temps et d'efforts. Le chapitre décrit le principe de chaque méthode et ses caractéristiques, l'accent a été mis sur la méthode de résolution qui sera élaborée dans ce projet.

Le dernier chapitre fournit un ensemble d'exemples détaillés de chaque propriété logique notamment la satisfaibilité, la validité, la compatibilité et la déduction. En plus, nous avons présenté tous les outils utilisés pour la réalisation de ce projet.

Chapitre I

Syntaxe et Sémantique Propositionnelle

1 Introduction

La logique propositionnelle, appelée aussi *logique booléenne*, est le type le plus simple des logiques mathématiques, elle est considérée comme une base au développement de la plupart des autres logiques. La logique booléenne permet l'étude des énoncés « des propositions » qui peuvent être soit vrais soit faux dans une situation donnée.

La logique propositionnelle permet de combiner de telles propositions, alors que les propositions se coordonnent et s'enchaînent afin de produire des raisonnements valides via des connecteurs logiques. Ainsi, elle permet de construire des énoncés plus complexes appelés *formules*.

Afin de donner une description pour un énoncé, la logique booléenne est définie par deux aspects, à savoir l'aspect *syntaxe* et *sémantique*. La syntaxe concerne la forme et la construction d'un énoncé, sa structure linguistique et comment l'écrire. L'aspect sémantique étudie la signification "sens" des unités linguistiques et de leur combinaison pour la détermination d'une valeur de vérité.

2 Syntaxe de la Logique Propositionnelle

La syntaxe d'un langage définit l'alphabet et les règles d'écriture (grammaire) des expressions du langage. Cet aspect ne s'intéresse pas au sens. [1]

2.1 Proposition

On appelle une proposition un énoncé qui peut être vrai ou faux. On dit alors que les deux valeurs de vérité d'une proposition sont « vrai » et « faux». [2]

Exemple: Le carré de réel est un réel positif.

Une proposition peut être :

- Atomique : une phrase indécomposable, ex : il pleut, il neige.
- Composé : se construit à partir des formules atomiques à l'aide des connecteurs logiques, ex : il pleut mais il ne neige pas.

2.2 Langage Propositionnel

2.2.1 Atomes

Nous appelons *atomes* ou *variables propositionnelles* ou *propositions élémentaires* des énoncés dont nous ne connaissons pas la structure interne, et qui gardent leur identité tout au long du calcul propositionnel qui nous occupe [3]. Un atome est représenté par une lettre (*a, b, p, q, . . .*) qui peut prendre une valeur de vérité (vrai ou faux). [4]

2.2.2 Littéral

Un littéral est un atome (littéral positif) ou la négation d'un atome (littéral négatif). [3]

Exemple : A est littéral.

¬A est littéral.

2.2.3 Clause

Une clause est une disjonction de littéraux ($p_1 \vee p_2 \vee \ldots \vee p_n$), les littéraux pouvant être positifs ou négatifs. [3]

Exemple : $(B \vee \neg C \vee Q)$

2.2.4 Connecteurs logiques

Les connecteurs sont des opérateurs qui permettent, en reliant deux propositions, de former une nouvelle proposition: [5]

Négation

La négation d'une proposition (vraie) est une proposition (fausse), par exemple si P est vraie alors ¬P est fausse. [3]

Notation : ¬

Quelques phrases ont le même sens :

- "non"
- "il est faux que".

Conjonction

On peut constater que la conjonction de deux propositions ($P \wedge Q$) est vraie lorsque les deux propositions sont vraies. Dans les autres cas elle est fausse. [3]

Notation : ∧

Quelques phrases ont le même sens :

- "mais" (sous-entendu mais aussi)
- "à la fois"
- "quoique"
- "bien que".

Disjonction

Définit comme étant vraie lorsque l'une des deux propositions ($P \vee Q$) est vraie et étant fausse si les deux propositions sont fausses. [3]

Notation : ∨

Quelques phrases ont le même sens :

- " ou "
- "à moins que"
- "Soit …soit".

Implication

Si P et Q sont deux propositions, on peut constater que l'implication " P implique Q " est fausse lorsque P est vrai et Q est faux. Dans les autres cas, elle est vraie. [3]

Notation : →

Quelques phrases ont le même sens :

- "si …alors"
- "implique"
- "est une condition nécessaire de"
- "a pour conséquence"
- " donc".

Equivalence

P et Q sont équivalents si les deux propositions ont la même valeur de vérité, on lit: P si et seulement si Q. La valeur de vérité de P↔Q est la valeur de vérité de (P→Q) ∧ (Q→P). [3]

Notation : ↔

Quelques phrases ont le même sens :

- "si et seulement si"
- "équivaut à"

2.3 Formule de la Logique propositionnelle

2.3.1 Formule propositionnelle

Les formules propositionnelles sont composées à partir des variables propositionnelles à l'aide des opérateurs logique [6].

- Toute variable est une FP
- Si A est une FP, ⌉A est une FP
- Si A et B sont des FP, il en est de même des expressions :
  (A) K (B) tel que K ∈ {∧,∨, →,⌉,↔}.

2.3.2 Formule bien formée

Une formule bien formée est une formule ou expression logique qui respecte les contraintes imposées par la syntaxe du langage. [7]

Les règles d'écriture

Les règles d'écriture précisent la manière dont sont assemblés les symboles de l'alphabet pour former des expressions bien formées (ou formules) du langage propositionnel: [1]

1. Tous les symboles de propositions sont des formules de LP.
2. Si R est une formule de LP, alors ⅂R est une formule de LP.
3. Si R et B sont des formules de LP, alors (R ∧ B), (R ∨ B), (R→B), et (R→B) sont des formules de LP.
4. Une expression n'est une formule que si elle est écrite conformément a`u règles 1, 2 et 3.

Exemple :

Non formule	**FBF**
P ∨	P
Q⅂ P	⅂Q
((A ∧ B) →) P	((A ∧ B) →P)

Table 1 : Exemple des formules bien formées

2.3.3 Sous formule propositionnelle

Les sous-formules d'une formule Q sont définies par : [8]

- Q est une SF de Q.
- Si ⅂Q' est une SF de Q alors Q' est une SF de Q.
- Si Q1 ∧ Q2 est une SF de Q alors Q1 et Q2 sont des SF de Q.
- Si Q1 ∨ Q2 est une SF de Q alors Q1 et Q2 sont des SF de Q.
- Si Q1 → Q2 est une SF de Q alors Q1 et Q2 sont des SF de Q.
- Si Q1 ↔ Q2 est une SF de Q alors Q1 et Q2 sont des SF de Q.

Exemple : Q= ((A) ∧(B)) →(C)

Donc ((A) ∧(B)) →(C), (A) ∧(B), C, A, B sont des sous-formules de Q.

2.4 Arbre de décomposition

Toute proposition appartient à un certain niveau, ce qui veut dire qu'elle a été construite progressivement à partir de propositions simples (SF), par ajouts successifs de connecteurs. L'arbre de décomposition d'une proposition est l'analyse qui permet de retracer l'histoire de la construction de cette proposition. [9]

La construction d'un arbre de décomposition est faite à travers les étapes suivantes :

- ✓ On découpe au niveau du connecteur principal d'une formule propositionnelle φ.
- ✓ On supprime les parenthèses les plus externes de chacun des arguments du connecteur principal.
- ✓ On répète l'opération de décomposition pour chacun des arguments jusqu'à arriver aux atomes

Exemple :

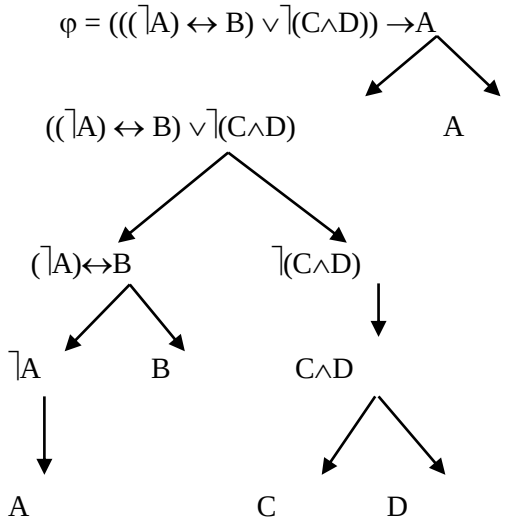


Figure 1 : Arbre de décomposition

2.5 Notation Polonaise

Les logiciens polonais ont proposé des notations permettant de se passer complètement des parenthèses. La notation polonaise directe, ou préfixée, consiste à écrire l'opérateur avant ses opérandes, la notation polonaise inverse, ou postfixée, consiste à écrire l'opérateur après ses opérandes. [10]

Notation infixée	((p →q) ↔ (¬q →¬p))	(p → (q ↔ ¬(q →¬p)))
Notation préfixée	↔→ p q →¬ q ¬ p	→p ↔ q ¬ →q ¬ p
Notation postfixée	p q → q ¬ p ¬→ ↔	p q q p ¬ → ¬ ↔ →

Table 2 : Notation Polonaise

2.6 Longueur et Complexité d'une Formule Propositionnelle

2.6.1 Longueur

La longueur d'une proposition est le nombre total de symboles de l'alphabet qui la composent. [9]

Exemple:

Q	**Long(Q)**
B	1
⌉(C∧D)	6
((A) ∧(B)) →(C))	14

Table 3 : Exemple de longueur

2.6.2 Complexité

La complexité d'une formule φ est le nombre d'opérateurs propositionnels qu'elle contient.

Exemple :

φ	**C(φ)**
B	0
⌉(C∧D)	2
((A) ∧(B)) →(C))	2

Table 4 : Exemple de complexité

3 Sémantique de la Logique propositionnelle

La sémantique détermine les règles d'interprétation des formules. On attribue des valeurs de vérité (vrai/faux) aux propositions élémentaires et on explique comment les connecteurs se comportent vis-à-vis de ces valeurs de vérité. On exprime souvent ce comportement par une table de vérité. Par conséquent, la sémantique d'un langage pour le calcul des propositions a pour but de donner une valeur de vérité aux formules du langage. Elle est aussi appelée la théorie des modèles. [1]

3.1 Valuation

On appelle valuation d'un ensemble de variables propositionnelles v(L), une fonction v de v(L) dans {0, 1}. [11]

$$(v : v(L) \rightarrow \{1,0\}).$$

Si v(L) contient n variables on a 2^n valuations possibles.

Exemple : V1(A)=0, V1(B)=1
V2(A)=1, V2(B)=1

	A	B
V1 →	0	1
V2 →	1	1
V3 →	0	0
V4 →	1	0

Table 5 : Valuations possibles pour deux atomes

3.2 Interprétation

L'interprétation $[p]_v$ d'une formule p par rapport à l'affectation de la valuation v est définie par : [6]

- $[x]_v = v(x)$
- $[\neg p]_v = \begin{cases} 0 \text{ si } [p]_v = 1 \\ 1 \text{ si } [p]_v = 0 \end{cases}$
- $[p \wedge q]_v = \begin{cases} 0 \text{ si } [p]_v = 0 \textbf{ ou } [q]_v = 0 \\ 1 \text{ si } [p]_v = 1 \textbf{ et } [q]_v = 1 \end{cases}$
- $[p \vee q]_v = \begin{cases} 0 \text{ si } [p]_v = 0 \textbf{ et } [q]_v = 0 \\ 1 \text{ si } [p]_v = 1 \textbf{ ou } [q]_v = 1 \end{cases}$
- $[p \rightarrow q]_v = \begin{cases} 0 \text{ si } [p]_v = 1 \textbf{ et } [q]_v = 0 \\ 1 \text{ les autres cas} \end{cases}$
- $[p \leftrightarrow q]_v = \begin{cases} 0 \text{ autres cas} \\ 1 \text{ si } [p]_v = 1 \textbf{ et } [q]_v = 1 \\ 1 \text{ si } [p]_v = 0 \textbf{ et } [q]_v = 0 \end{cases}$

Exemple : La fonction d'évaluation des variables est : V= {(p, V), (q, F), (r, V), (s, V)}
La formule: $(p \vee s) \leftrightarrow (s \wedge q)$

$V(p) = 1$

$V(s) = 1$

$[p \vee s]_v = 1$

$V(q) = 0$

$[s \wedge q]_v = 0$

$[(p \vee s) \leftrightarrow (s \wedge q)]_v = 0$

3.3 Analyse sémantique d'une formule

Il existe plusieurs méthodes pour l'analyse de la valeur de vérité d'une formule propositionnelle pour toutes les valuations possibles des variables, la plus simple c'est la table de vérité.

3.3.1 Table de vérité

Une table de vérité d'une formule φ est un tableau qui représente la valeur de φ pour toutes les valeurs possibles des variables de φ.

Chaque ligne de la table de vérité définit une assignation pour les variables des formules qui sont présentées dans les colonnes de la table et chaque colonne donne la valeur d'une formule. [12]

Exemple :

P	**Q**	**¬p**	**P ∧ Q**	**P∨Q**	**P→Q**	**P↔Q**
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Table 6: table de vérité

3.4 Équivalence des formules bien formées

Deux formules sont équivalentes quand elles ont la même valeur dans toute interprétation (notation : A ≡ B). [11]

1. Par commutativité :
   - |= [(A∧B) ≡ (B∧A)]
   - |= [(A∨B) ≡ (B∨A)]
   - |= [(A ≡ B) ≡ (B ≡ A)]
2. Par associativité :
   - |= [((A∧B)∧C) ≡ ((A∧(B∧C))]
   - |= [((A∨B)∨C) ≡ ((A∨(B∨C))]
   - |= [((A ≡ B) ≡ C) ≡ ((A ≡ (B ≡ C))]
3. Par distributivité
   - |= [((A∧B)∨C) ≡ ((A∨C)∧(B∨C))]

- |= [((A∨B)∧C) ≡ ((A∧C)∨(B∧C))]

4. Par absorption :
   - |= [A∧(A∨B)) ≡ A]
   - |= (A∨(A∧B)) ≡ A]
5. Par idempotence
   - |= [(A∧A) ≡ A]
   - |= [(A∨A) ≡ A]
6. Par les lois de Morgan :
   - |= [¬(A∧B) ≡ (¬A∨¬B)]
   - |= [¬(A∨B) ≡ (¬A∧¬B)]
7. Par implication :
   - |= [(A → B) ≡ (¬A∨B)]
   - |= [(A → B) ≡¬(A∧¬B)]
8. Par équivalence :
   - |= [(A ≡ B) ≡ ((A → B)∧(B →A))]
   - |= [(A ≡ B) ≡ ((A∧B)∨(¬A∧¬B))]
9. Par contraposition :
   - |= [(A → B) ≡ (¬B →¬A)]
10. Par auto distributivité :
    - |= [(A → (B → C)) ≡ ((A → B) → (A → C))]
11. Par import-export :
    - |= [(A → (B → C)) ≡ (B → (A → C))]
12. Par transitivité
    - |= [((A → B)∧(B →C)) ≡ (A →C)]

3.5 Forme normale d'une formule propositionnelle

3.5.1 Les Types des Formes Normales

Les formes normales sont une manière de représenter une formule par une autre formule dont les connecteurs logiques sont restreints et ordonnés selon un certain ordre. Les formes normales les plus connues sont : [11]

- FNN : Forme Normale de Négation
- FNC : Forme Normale Conjonctive
- FND : Forme Normale Disjonctive

Forme Normale de Négation

Une formule est en forme normale de négation si [16] :

- Elle est construite avec les connecteurs ¬, ∧ et ∨, seulement ;
- Elle ne contient pas d'applications de l'opérateur ¬ sauf des applications devant les variables propositionnelles.

Exemple :

- ¬x ∧y ∧ z et (¬x ∨ y) ∧ (¬z ∨ x) sont en forme normale de négation
- ¬ (x∧y) et ¬¬y ne le sont pas

Forme Normale Disjonctive

Une formule est en forme normale disjonctive (FND) si et seulement si c'est une disjonction de conjonctions de littéraux, c'es-a-dire une formule de la forme: [13]

$$\vee_i (\wedge_j(\neg)\, p_{i,j})$$

Exemple: (¬A∧¬B) ∨ (¬C∧D) ∨ E.

Forme Normale Conjonctive

Une formule est en forme normale conjonctive (FNC) si et seulement si c'est une conjonction de disjonctions de littéraux, c'es-a-dire une formule de la forme : [13]

$$\wedge_i (\vee_j(\neg)\, p_{i,j})$$

Exemple: P∧ (Q∨¬R) ∧ (¬P ∨ R)

3.5.2 La transformation en Formes Normales

En pratique, on utilise les transformations successives suivantes pour obtenir les formes normales : [13]

1. Elimination des connecteurs → et ↔ grâce aux équivalences suivantes :

(φ → ψ) ≡ (¬φ ∨ ψ)
(φ ↔ ψ) ≡ (¬φ ∨ ψ) ∧ (φ ∨¬ψ)

2. Entrer les négations le plus à l'intérieur possible :

¬ (φ ∧ ψ) ≡ (¬φ ∨¬ψ)
¬ (φ ∨ ψ) ≡ (¬φ ∧¬ψ)

3. Utilisation des distributivités de ∧ et ∨ l'un par rapport à l'autre :

$(\varphi \wedge (\psi \vee \chi)) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \chi)$
$(\varphi \vee (\psi \wedge \chi)) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \chi)$

Voici quelques exemples expliquant comment appliquer les règles précédentes

Exemple1 : ¬(X∧(Y∨¬Z))

- se transforme en ¬X ∨ ¬(Y∨¬Z)
- se transforme en ¬X ∨ (¬Y∧Z) ⇒ la formule est sous FNN
- se transforme en ¬X ∨ (¬Y∧Z) ⇒ la formule est sous FND

exemple 2 : A ∧(¬ (B ∨C)∧ ¬¬(D ∨P))

- Se transforme en A ∧ (¬(B ∨C)∧(D ∨ P))
- Se transforme en A ∧ ((¬B ∧¬C)∧(D ∨ P)) ⇒ la formule est sous FNN
- Se transforme en A ∧ (((¬B ∧¬C)∧ D)∨((¬B ∧¬C)∧ P))
- Se transforme en A ∧ ((¬B ∧¬C ∧ D)∨(¬B ∧¬C ∧ P))
- Se transforme en (A ∧¬B ∧¬C ∧ D)∨(A ∧¬B ∧¬C ∧ P) ⇒ la formule est sous FND

3.6 Satisfiabilité et Validité d'une formule

3.6.1 Formule satisfaite

Une formule satisfiable ou sémantiquement consistante est une formule vraie dans au moins une interprétation. [11]

$$\varphi \text{ satisfaite} \leftrightarrow \exists V\ [\varphi]_v = 1.$$

Exemple : x ∨ y

X	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

Table 7: Exemple d'une formule satisfaite

3.6.2 Formule Insatisfaite

Une formule insatisfaite, ou sémantiquement inconsistante, est une formule fausse dans toute interprétation. [11]

$$\varphi \text{ insatisfaite} \leftrightarrow \forall V\ [\varphi]_v = 0.$$

Exemple : $y \wedge \neg y$

y	y	$y \wedge \neg y$
0	1	0
0	1	0
1	0	0
1	0	0

Table 8: Exemple d'une formule insatisfaite

3.6.3 Formule Valide

Une formule valide est une formule φ vraie quelles que soient les valeurs de vérité des atomes qui la composent. On la note $\models \varphi$. [11]

$$\varphi \text{ valide} \leftrightarrow \forall V\ [\varphi]_v = 1.$$

Exemple : $\varphi = (x \rightarrow y) \leftrightarrow (\neg x \vee y)$

X	Y	$\neg x$	$x \rightarrow y$	$\neg x \vee y$	$(x \rightarrow y) \leftrightarrow (\neg x \vee y)$
0	0	1	1	1	1
0	1	1	1	1	1
1	0	0	0	0	1
1	1	0	1	1	1

Table 9: Exemple d'une formule valide

3.6.4 Formule Invalide

Une formule invalide est fausse dans au moins une interprétation. [11]

$$\varphi \text{ invalide} \leftrightarrow \exists V\ [\varphi]_v = 0.$$

Exemple : $A \vee B$

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

Table 10: Exemple d'une formule invalide

3.7 Modèle

3.7.1 Modèle d'une formule propositionnelle

Un modèle de formule φ est une valuation v telle que $[\varphi]_v = 1$. [14]

On note le modèle de φ: $V \vDash \varphi \leftrightarrow [\varphi]_v = 1$.

Exemple : $\varphi = p \vee q \rightarrow p \wedge q$ a deux modèles :

$V1 \vDash \varphi \setminus V1(p)=0$ et $V1(q)=0$

$V2 \vDash \varphi \setminus V2(p)=1$ et $V2(q)=1$

3.7.2 Modèle d'un ensemble de formules propositionnelles

Un modèle V d'un ensemble de formules Σ si et seulement si V est un modèle pour toute formule φ dans Σ. [15]

Notation : $V \vDash \Sigma \leftrightarrow \forall \varphi \in \Sigma \quad [\varphi]_v = 1$. [16]

Exemple: $\Sigma = \{p \vee q, p \rightarrow q \vee r, r\}$ Σ a un modèle.

$V1 \vDash \Sigma$ / $V1(p)=1, V1(q)=0, V1(r)=1$

Ci-dessous un exemple détaillé sur les modèles d'une formule et les modèles d'un ensemble de formules.

	A	**b**	**c**	**a∧¬c**	**c ∨ b**
V1	0	0	0	0	0
V2	0	0	1	0	1
V3	0	1	0	0	1
V4	0	1	1	0	1
V5	1	0	0	1	0
V6	1	0	1	0	1
V7	1	1	0	1	1
V8	1	1	1	0	1

Table 11: Modèles pour les formules

- Les modèles de **a ∧¬c** sont : **V5, V7**
- Les modèles de **c ∨ b** sont : **V2, V3, V4, V6, V7, V8**
- Le modèles de l'ensemble **{a∧¬c, c ∨ b}** est : **V7**

3.8 Compatibilité

3.8.1 Compatibilité d'une formule

Une formule propositionnelle φ est compatible si et seulement si φ a au moins un modèle. La formule φ est compatible donc φ est satisfiable. [16]

φ est compatible ↔ ∃ V $[\varphi]_V$ =1. Ex: x ∨ y.

X	Y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

Table 12: Compatibilité d'une formule

φ est incompatible ↔ ∀ V $[\varphi]_V$ =0. Ex: p ∧ ⅂p.

P	⅂P	p ∧ ⅂p
0	1	0
0	1	0
1	0	0
1	0	0

Table 13: incompatibilité d'une formule

3.8.2 Compatibilité d'un ensemble de formules

Un ensemble de formules Σ est compatible si et seulement si il a au moine un modèle. [16]

Σ est compatible ↔ ∃ V ∀ φ ∈ Σ $[\varphi]_v$ =1.

Σ est incompatible ↔ ∀ V ∃ φ ∈ Σ $[\varphi]_v$ =0.

Exemple : Σ = {A→B, A ∧ ⅂C, C ∨ B} est compatible pour V(A)=1, V(B)=1, V(C)=0.

	A	**B**	**C**	**A→ B**	**A∧⅂C**	**C ∨ B**
V1	0	0	0	1	0	0
V2	0	0	1	1	0	1
V3	0	1	0	1	0	1
V4	0	1	1	1	0	1
V5	1	0	0	0	1	0
V6	1	0	1	0	0	1
V7	1	1	0	1	1	1
V8	1	1	1	1	0	1

Table 14: compatibilité d'une formule

Exemple : $\Sigma = \{A \rightarrow C, A \wedge \neg C, C \vee B\}$ est incompatible.

	A	B	C	A→C	A∧¬C	C∨B
V1	0	0	0	1	0	0
V2	0	0	1	1	0	1
V3	0	1	0	1	0	1
V4	0	1	1	1	0	1
V5	1	0	0	0	1	0
V6	1	0	1	1	0	1
V7	1	1	0	0	1	1
V8	1	1	1	1	0	1

Table 15: incompatibilité d'une formule

Aucun modèle de l'ensemble.

3.9 Conséquence Logique

3.9.1 Conséquence d'une formule

Une formule f1 est conséquence logique d'une formule f2 si et seulement si tout modèle d'une f2 est un modèle de f1et on note f2 ⊨ f1. [17]

$$f1 \text{ et conséquence } f2 \leftrightarrow (\forall V ([f2]_v = 1 \rightarrow [f1]_v = 1)). \text{ [16]}$$

Exemple :

$\Omega = A \wedge B$	$\varphi = A \vee B$	φ est conséquence de Ω
$\Omega = A \wedge (A \rightarrow B)$	$\varphi = A \wedge B$	φ est conséquence de Ω

3.9.2 Conséquence d'un ensemble de formules

Une formule f est conséquence logique d'un ensemble de formules Σ (noté Σ⊨f) si tout modèle de Σ est un modèle de f. [8]

$$(\Sigma \models \varphi) \leftrightarrow \forall V ((\forall \varphi i \in \Sigma\ [\varphi i]_v = 1) \rightarrow [\varphi]_v = 1). \text{ [16]}$$

Exemple :{A→B, A∧¬C, C ∨ B} ⊨ A ∧ B

A	B	C	A→B	A∧¬C	C ∨ B	A ∧ B
0	0	0	1	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	1	0	0	1	0
1	1	0	1	1	1	1
1	1	1	1	0	1	1

Table 16: Conséquence d'un ensemble de formules

Exemple : A ∧ B n'est pas conséquence de {A→B, C ∨ B}.

A	B	C	A→B	C ∨ B	A ∧ B
0	0	0	1	0	0
0	0	1	1	1	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	1	1	1

Table 17: F n'est pas conséquence de Σ de formule

4 Conclusion

Dans ce chapitre, nous avons traité les deux concepts de base de la logique propositionnelle, où nous avons fourni une explication détaillée à toutes les parties de l'aspect syntaxique, à savoir les alphabets du langage propositionnel tels que les constantes, les atomes, les connecteurs logiques, etc. Puis, nous avons étudié la partie sémantique de la logique propositionnelle en discutant l'assignation d'une valeur de vérité à une formule à l'aide des exemples clairs et détaillés.

Nous concluons de ce qui précède que de nombreux problèmes peuvent être formulés en logique des propositions, mais quand le nombre de variables est important ou inconnu, l'utilisation des tables de vérité devient difficile et parfois impossible. Dans le deuxième chapitre, nous allons présenter une méthode formelle (qui ne fait pas appel aux tables de vérité) pour montrer que les formules sont des tautologies ou des conséquences logiques pour d'autres formules.

Chapitre II

Méthodes de Preuve

1 Introduction

La logique propositionnelle sert à étudier la valeur de vérité d'une formule pour toutes les valuations possibles des variables qu'elle contient, cela afin de prouver certaines propriétés. Bien qu'il soit possible d'énumérer toutes les évaluations pour créer une table de vérité, la méthode est coûteuse en espace et en temps, surtout lorsque le nombre de variables est important ou parfois inconnu. Dans ce chapitre, nous présentons quelques méthodes de preuve en logique propositionnelle qui aide ceci.

2 Méthodes de Preuve

C'est une approche systématique pour établir si une déclaration, une logique donnée, est correcte ou non. Tandis que, il serait complet s'il permettait la validation de toutes les instructions correctes dans la logique correspondante. Et il serait valide si toutes les instructions déclarées comme valides étaient réellement valides dans la logique correspondante. [18]

Nous essayons de présenter quelques méthodes de preuves qui concerne la logique booléenne tels que la méthode des tableaux sémantiques, la méthode des arbres en propositionnel, et la méthode de résolution qui sera détaillée dans le troisième chapitre.

2.1 Méthode des Tableaux Sémantiques

La méthode des tableaux sémantiques consiste en la recherche systématique d'un modèle d'une formule φ donnée. Le fait d'imposer une valeur de vérité à une formule peut déterminer univoquement la valeur des composants de la formule, ou au contraire laisser plusieurs choix possibles. Cette méthode conduit à la construction progressive d'une arborescence spécifique appelée table sémantique. [10]

Exemple :

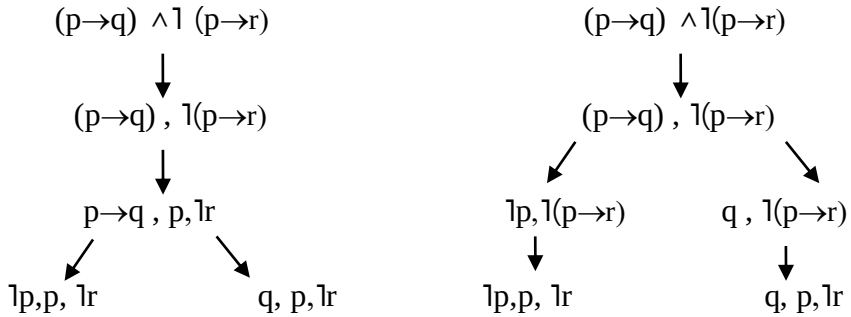


Figure 2: Deux tableaux sémantiques pour la même formule.

Selon l'ordre dans lequel les règles de construction sont appliquées, la formule peut conduire à plusieurs tables sémantiques différentes, mais tous conduisent au même résultat concernant la consistance de la formule. La signification commune des deux tableaux de la figure est une interprétation, est un modèle de la formule (p→q) ∧¬ (p→r) si et seulement si c'est un modèle de l'un des ensembles {¬p, p, ¬r}, {q, p, ¬r}. [10]

2.1.1 Construction des Tableaux Sémantiques

La construction des tableaux sémantiques est basée sur la division des formules en trois catégories :

- les littéraux.
- les formules conjonctives.
- les formules disjonctives.

La construction est basée sur deux types de règles de décomposition de formules : les règles de *prolongation* (type α) qui utilisées pour les formules conjonctives et les règles de *ramification* (type β) pour les formules disjonctives. [10]

α	**α_1**	**α_2**
$A_1 \wedge A_2$	A_1	A_2
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$
$\neg(A_1 \rightarrow A_2)$	A_1	$\neg A_2$
$\neg(A_2 \rightarrow A_1)$	$\neg A_1$	A_2
$A_1 \leftrightarrow A_2$	$A_1 \rightarrow A_2$	$A_2 \rightarrow A_1$

β	**β_1**	**β_2**
$B_1 \vee B_2$	B_1	B_2
$\neg(B_1 \wedge B_2)$	$\neg B_1$	$\neg B_2$
$B_1 \rightarrow B_2$	$\neg B_1$	B_2
$B_2 \rightarrow B_1$	B_1	$\neg B_2$
$\neg (B_1 \leftrightarrow B_2)$	$\neg (B_1 \rightarrow B_2)$	$\neg (B_2 \rightarrow B_1)$

Table 18: Règles de décomposition de type α et β.

2.1.2 Preuve de la Validité d'une formule

Prouver que φ est valide revient à prouver que ¬φ est non satisfaite.

φ Valide ⇔¬φ est non satisfaite. [16]

Exemple : $\varphi = A \vee \neg A$ valide ?

$$\varphi' = \neg(A \vee \neg A)$$

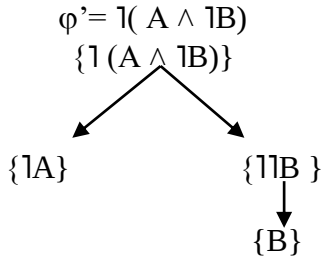
$$\{\neg (A \vee \neg A)\}$$

$$\{\neg A, \neg (\neg A)\}$$

$$\{\neg \underline{A}, \underline{A}\}$$

φ'non satisfaite ⇒ $\varphi = A \vee \neg A$ valide

Exemple : φ = A ∧ ٦B

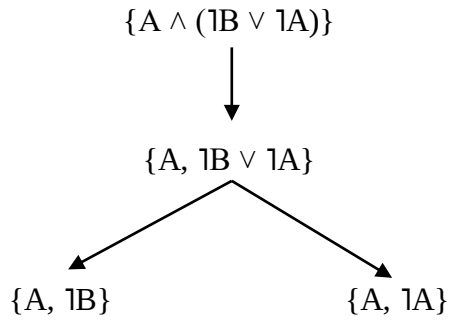


φ' est satisfaite ⇒ (A ∧ ٦B) est invalide

2.1.3 Preuve de la satisfiabilité

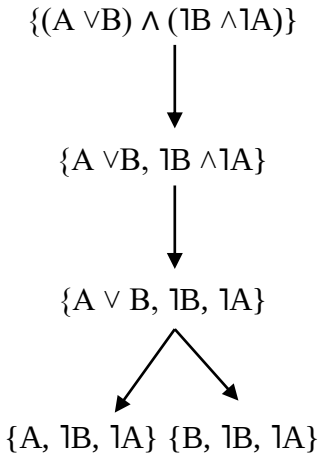
Pour prouver qu'une formule propositionnelle φ est satisfaite on cherche systématiquement un modèle. On suppose que la formule φ est vraie et on cherche un modèle V. [16]

Exemple : φ = A ∧ (٦B ∨ ٦A)



V(A)=1 et V(B)=0 Donc A ∧ (٦B ∨ ٦A) est satisfaite.

Exemple : φ = (A ∨B) ∧ (٦B ∧٦A)



(A ∨B) ∧ (¬B ∧¬A) est non satisfaite.

2.2 Méthode des Arbres en Propositionnel

La méthode des arbres est une méthode de preuve indirecte du caractère tautologique (ou non) d'une formule. On cherche à déterminer si sa négation peut être vraie pour au moins une interprétation des lettres de proposition, non pas si est vraie (ou non) pour toute interprétation des lettres de proposition (table de vérité). [19]

2.2.1 Principe de la Méthode

La méthode des arbres permet de trouver toutes les interprétations pour lesquelles une formule est vraie, à la fin de la procédure, si aucune explication n'est présentée qui rend la formule correcte, nous pouvons conclure que la formule est contradictoire : sa négation est alors tautologique.

Si la formule est vraie pour une interprétation, alors ses sous-formules directes, ou leur négation, doivent elles-mêmes être vraies pour la même interprétation. Pour que l'on arrive, pas à pas, aux formules élémentaires ou à leur négation qui doit être vraie pour la même interprétation pour que la forme de départ soit correcte. [19]

2.2.2. Etudier la Tautologie

Afin de comprendre bien la méthode des arbres en propositionnel, nous introduisons un exemple pour traiter la propriété de la validité de la formule : $(p \vee q) \rightarrow ((r \rightarrow p) \wedge (r \vee q))$.

Le principe de cette méthode est d'étudier la négation de la formule comme suit :

Exemple :

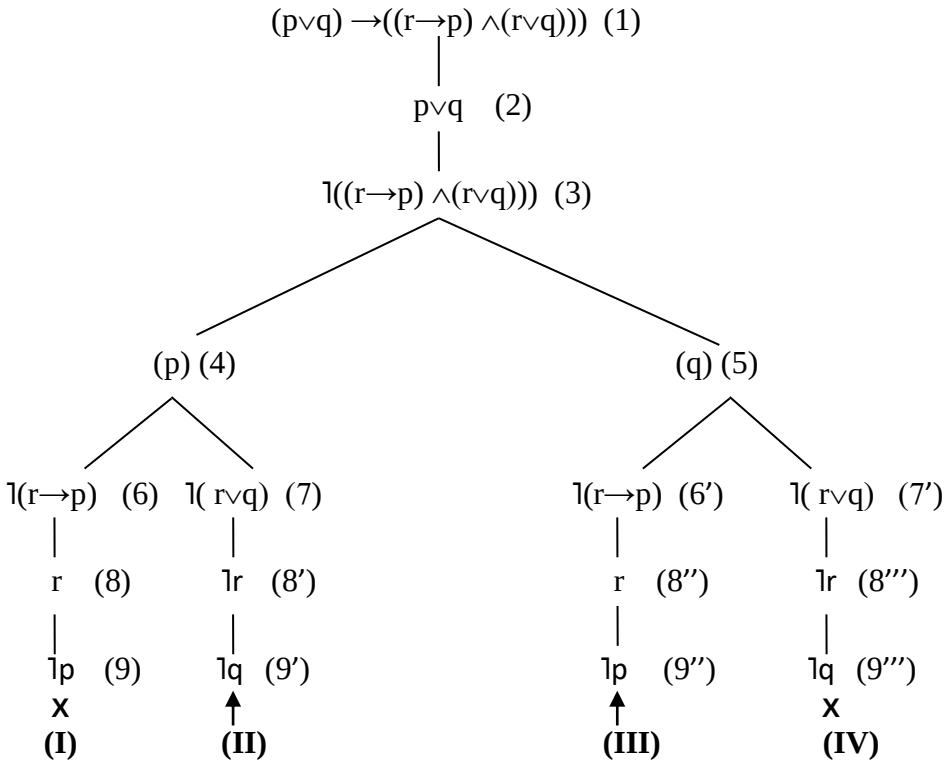


Figure 3 : Exemple 1 d'un Arbre en propositionnel

Règles utilisées :

- (2), (3) sont obtenus à partir de (1) par la règle "¬→"
- (4), (5) son obtenus à partir de (2) par la règle "∨"
- (6), (7) et (6′), (7′) sont obtenus à partir de (3) par la règle "¬∧"
- (8), (9) et (8′′), (9′′) sont obtenus à partir de (6) et (6′) respectivement, par la règle "¬→"
- (8′), (9′) et (8′′′), (9′′′) sont obtenus à partir de (7) et (7′) respectivement, par la règle "¬∨"

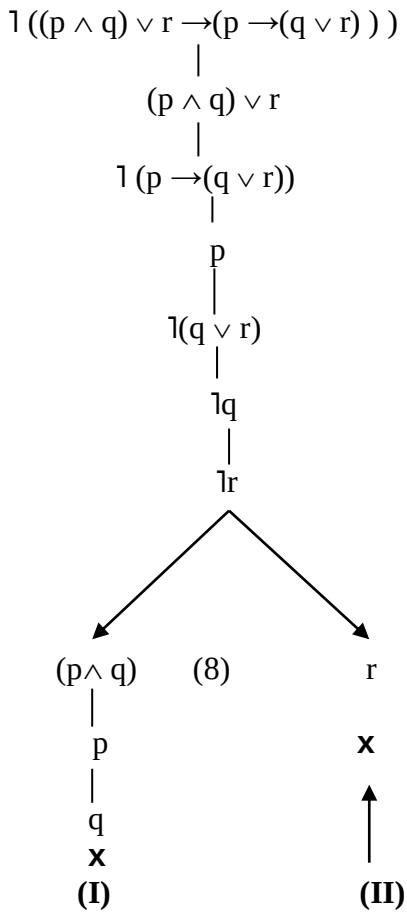
A partir de l'arbre, on constate que les branches (II) et (III) sont ouvertes : il y a donc deux interprétations V1 et V2 qui satisfont la négation de la formule initiale, à savoir :

– branche (II) : V1 (p,q,r) = (1,0,0)

– branche (III) : V2 (p,q,r) = (0,1,1)

Donc, la formule pour laquelle on a fait l'arbre n'est pas tautologique.

Exemple : $Q = (p \wedge q) \vee r \rightarrow (p \rightarrow (q \vee r))$



Etape 1 — Sur (1), règle ¬ →

Etape 2 — Sur (3), règle¬ →

Etape 3 — Sur (5), règle ¬∨

Etape 4 — Sur (2), règle ∨

Etape 5 — Sur (8), règle ∧

Figure 4 : Exemple 2 d'un Arbre en propositionnel

Sur chaque branche, apparaissent une lettre de proposition (atome) et la même lettre précédée du symbole de la négation (sa littéral complémentaire). Sur la branche (I), on trouve q et ¬q ; sur la branche (II), on trouve r et ¬ r. La formule φ est donc tautologique car sa négation (la formule initiale) n'est pas satisfiable.

2.3 Méthode de Résolution

La méthode de résolution est la technique de preuve la plus souvent utilisée dans les programmes de démonstration automatique. Cette méthode traite les formules par réfutation : elle permet de démontrer la validité d'une formule en démontrant l'inconsistance de sa négation. Nous pouvons appliquer la méthode de résolution à n'importe quel ensemble de formules mais, en ce qui concerne les méthodes que nous avons vues précédemment, il est plus facile d'en limiter à un certain type spécifique des formules, elles sont appelées formes clausales. [10]

2.3.1 Notion de Clause

Une clause est une formule bien formée qui a la forme d'une disjonction de littéraux, cas particulier : un littéral isolé est une clause. [22]

Exemple : $A \vee B \vee C \vee \neg D$, $\neg D$ sont des clauses.

2.3.2 Ensemble des Clauses

L'ensemble des clauses est compris comme la conjonction des clauses, donc c'est une formule en forme normale conjonctive. [23]

Exemple : $\{p \vee (\neg q) \vee s\ ,\ p \vee q \vee (\neg s)\ ,\ s \vee \neg t\} \Rightarrow (p \vee (\neg q) \vee s) \wedge (p \vee q \vee (\neg s)) \wedge (s \vee \neg t)$

2.3.3 Clause vide

Une clause C ne contenant aucun littéral est appelée clause vide. Elle est notée : $C = \varnothing$. [25]

2.3.4 Clause Résolvante

Si C1 et C2 sont deux clauses, la clause C1 contient p1 et C2 contient p2 tel que $p1 = \neg p2$. La clause C est appelée *résolvante* ou *résolvant de C1 et C2* dont les clauses sont la disjonction de ceux de C1 moins p1 et ceux de C2 moins p2. [23]

Exemple :

- Le résolvant de $C1 = p \vee q \vee s \vee \neg t$ et $C2 = u \vee (\neg q) \vee b \vee \neg t$
  est $C = p \vee s \vee (\neg t) \vee u \vee b$
- Le résolvant de $C1 = A \vee B$ et $C2 = \neg A \vee \neg B$
  est $C = A \vee \neg A$ **ou** $C = B \vee \neg B$
- Le résolvant de $C1 = \neg t$ et $C2 = t$
  est $C = \varnothing$ (l'ensemble vide)

Noter que chaque clause résolvante d'un ensemble Σ est une conséquence logique de Σ.

Démonstration:

Nous avons la règle d'inférence ci-dessous, tels que : A, B sont deux clauses et X un littéral.

$$\{\underline{X \vee A}, \underline{\neg X \vee B}\} \vdash \underline{A \vee B}$$

C1 C2 C clause résolvante

On sait que : $(\neg X \vee B) \leftrightarrow (\mathbf{X \rightarrow B})$
Et que $(A \vee X) \leftrightarrow (\neg \mathbf{A \rightarrow X})$
On a ainsi: $\{\neg A \rightarrow X, X \rightarrow B\}$, qui donne par règle du syllogisme: $\neg A \rightarrow B$,
$\neg A \rightarrow B$ est équivalent à $A \vee B$. [21]

Cette règle très simple est appelée *règle de résolution*.

Soient deux clauses C1 et C2 ; un résolvant C de C1, C2 est une conséquence logique de C1 et C2.

2.3.5 Méthode de résolution appliquée à un ensemble de clauses

Elle consiste à partir d'un Σ de clauses et à appliquer la règle de résolution un certain nombre de fois, en enrichissant chaque fois l'ensemble de départ avec les déductions qui sont faites, jusqu'à obtenir la conclusion désirée. [24]

Exemple : montrer la déduction suivante :

$$\{p \vee q \vee r, \neg p \vee q \vee r, \neg q \vee r\} \vDash r$$

Ensemble de départ: $\{p \vee q \vee r, \neg p \vee q \vee r, \neg q \vee r\}$

1er pas : on choisit $C1 = p \vee q \vee r$ $C2 = \neg p \vee q \vee r$
$C = q \vee r$
$\Rightarrow \{p \vee q \vee r, \neg p \vee q \vee r, \neg q \vee r, q \vee r\}$

2eme pas : on choisit $C1 = \neg q \vee r$ $C2 = q \vee r$
$C = r$
$\Rightarrow \{p \vee q \vee r, \neg p \vee q \vee r, \neg q \vee r, q \vee r, r\}$

Stop! On a trouvé r.

2.3.6 Résolution avec Réfutation

Le principe de résolution par réfutation est formé d'une unique règle d'inférence. Sa grande simplicité de mise en œuvre en fait une méthode très utilisée. [21]

Un ensemble de clauses est inconsistant (insatisfiable) si et seulement si l'ensemble vide est une conséquence valide de cet ensemble. On appelle *réfutation* la déduction de la clause vide à partir d'un ensemble. [20]

Donc à partir d'un Σ de clauses contenant la négation de la proposition à démontrer et appliquer la règle de résolution un certain nombre de fois, en enrichissant chaque fois l'ensemble de départ avec les déductions qui sont faites, jusqu'à obtenir ∅ si possible. [24]

Exemple :

Montrer que cet ensemble est inconsistant : {p∨q∨r, ¬p∨q∨r, ¬q∨r, ¬r}

C1= ¬q∨r hypothèse
C2= ¬r hypothèse
C3= ¬q résolvant (C1,C2)
C4= p∨q∨r hypothèse
C5= ¬p∨q∨r hypothèse
C6= q∨r résolvant (C4,C5)
C7= r résolvant (C6,C3)
C8= ∅ résolvant (C7,C2)

Stop! On a trouvé ∅ ⇒ {p∨q∨r, ¬p∨q∨r, ¬q∨r, ¬r} est inconsistant.

2.3.7 Algorithme de Réfutation

On peut appliquer la méthode de résolution par réfutation sur des formules propositionnelles, on sait que pour montrer qu'une formule bien formée φ est valide :

- C'est équivalent à montrer que ¬φ est inconsistante.
- C'est aussi équivalent à monter que l'ensemble des clauses de φ est insatisfiable.
- C'est aussi équivalent à monter qu'il existe une déduction de la clause vide ∅.

Début
Ecrire la négation d'une formule φ;
Mettre ¬φ sous forme d'un ensemble de clauses ;
Tant que la clause vide n'est pas rencontrée et qu'il existe des paires réductibles **faire**
Chercher des clauses résolvantes ;
Ajouter ce résultat à la liste des clauses ;
FTQ ;
Si on trouve la clause vide **alors**
φ est valide
Sinon
φ est non valide
Finsi ;
Fin; [22]

Exemple :

Montrer $\vdash \varphi$ tel que : $\varphi = (a \rightarrow (b \rightarrow r)) \rightarrow ((a \rightarrow b) \rightarrow (a \rightarrow r))$

Afin de montrer la tautologie d'une formule φ en utilisant la méthode de résolution avec réfutation, on montre la déduction de l'ensemble vide à partir de l'ensemble des clauses de sa négation (la formule φ initiale).

1. Mettre $\neg\varphi$ sous FNC :

$\neg \varphi = \neg ((a \rightarrow (b \rightarrow r)) \rightarrow ((a \rightarrow b) \rightarrow (a \rightarrow r)))$

$\neg \varphi = (a \rightarrow (b \rightarrow r)) \wedge \neg (a \rightarrow b \rightarrow (a \rightarrow r))$

$\neg \varphi = (\neg a \vee (b \rightarrow r)) \wedge (a \rightarrow b \wedge \neg (a \rightarrow r))$

$\neg \varphi = (\neg a \vee \neg b \vee r) \wedge ((\neg a \vee b) \wedge (a \wedge \neg r))$

2. Extraction des clauses :

$\Sigma = \{ \neg a \vee \neg b \vee r , \neg a \vee b, a, \neg r \}$

C1= $\neg a \vee \neg b \vee r$	hypothèse
C2= $\neg r$	hypothèse
C3= $\neg a \vee \neg b$	résolvant (C1,C2)
C4= a	hypothèse
C5= $\neg b$	résolvant (C3,C4)
C6= $\neg a \vee b$	hypothèse
C7= $\neg a$	résolvant (C5,C6)
C8= $\varnothing$	résolvant (C4,C7)

Stop! On a trouvé $\varnothing \Rightarrow \neg\varphi$ est insatisfiable $\Rightarrow \vdash \varphi$

3 Conclusion

Ce chapitre est présenté quelques méthodes de preuve qui permettent d'économiser l'espace mémoire et de réduire le temps de réponse, et il s'est concentré sur la méthode de résolution que nous avons adopté dans notre travail. Nous avons parlé de comment l'appliquer au domaine logique et nous avons présenté ses caractéristiques les plus importantes. Dans le chapitre suivant, nous allons essayer de clarifier et détailler cette méthode en montrant les différentes propriétés des formules par plusieurs exemples.

Chapitre III

Implémentation

1 Introduction

Dans ce chapitre, nous montrons l'outil qui a été créé pour étudier les propriétés des formules propositionnelles en présentant les différentes interfaces avec des exemples détaillés. Chaque interface explique son but et comment l'utiliser, en plus nous décrivent la plupart des outils et les langages de programmation que nous avons utilisés à la phase d'implémentation.

2 Les Objectifs

Le but de notre application est d'étudier les formules logiques et de fournir des résultats précis et non ambigus. Notre outil vérifie d'abord l'exactitude des formules propositionnelles dont leur écriture, puis les étudie en utilisant le principe de la méthode de résolution par réfutation.

3 Logiciels et matériels utilisés

Dans le domaine de la programmation et de l'informatique, en gardant à l'esprit que concevoir une application, fournir un environnement de programmation spécifique. Nous allons donc présenter l'ensemble des outils, programmes et langages qui ont aidé à construire notre outil.

3.1 Logiciels

3.1.1 Netbeans

Netbeans est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2 (Common Development and Distribution License). En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, **refactorisation**, éditeur graphique d'interfaces et de pages Web). Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle Java). [26]

3.1.2 Java

C'est un langage de programmation orienté objet, développé par Sun Microsystems. Il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitation (Windows, Linux, Macintosh, Solaris). Java donne aussi la possibilité de développer des programmes pour téléphones portables et assistants personnels. Enfin, ce langage peut-être utilisé sur internet pour des petites applications intégrées à la page web (applet) ou encore comme langage serveur (jsp). [27]

3.1.3 JDK

Le JDK (Java Development Kit) est un environnement de développement logiciel utilisé pour développer des applications et des applets Java. Il comprend le Java Runtime Environment (JRE), un interpréteur/chargeur (java), un compilateur (javac), un archiveur (jar), un générateur de documentation (java doc) et d'autres outils nécessaires au développement Java. [28]

3.1.4 JavaFX

Javafx est une bibliothèque graphique intégrée dans le JRE et le JDK de Java. Oracle la décrit comme « The Rich Client Platform », c'est-à-dire qu'elle permet de réaliser des interfaces graphiques évoluées et modernes grâce à de nombreuses fonctionnalités, telles que les animations, les effets, la 3D, l'audio, la vidéo, etc. Elle a de plus l'avantage d'être dans le langage Java, qui permet de réaliser des architectures avec des paradigmes objet, et aussi de pouvoir utiliser le typage statique. [29]

3.2 Matériels

Notre application est implémentée sur Lap top Lenovo avec les spécifications suivantes :

- Processeur: Intel(R) Core(TM) i5-3320MCPU @ 2.60GHz.
- RAM: 8Go.
- Système d'exploitation: Windows 10.
- Type du Système: Système d'exploitation 64, processeur x64.

4 Teste d'application et résultats

Dans cette section, nous essayons de présenter les différentes interfaces de notre système à travers des captures d'écran avec quelques exemples pour expliquer le fonctionnement de chaque partie de notre outil.

4.1 Interface principale

Le premier accès au notre application nommée « Résolution », tombe à l'interface ci-dessous « le démarrage ». Ce dernier contient un titre significatif "Résolution par réfutation, logique propositionnelle " et deux boutons principaux, le bouton Démarrer, par lequel on accède au traitement, et le bouton "Sortir".

Figure 5 : L'interface principale.

Lorsque vous appuyez sur le bouton Démarrer, l'application vous fournit l'interface des options qui contient quatre boutons, chaque un à son rôle : preuve de la validité, preuve de non satisfiabilité, preuve de compatibilité d'un ensemble de formules et preuve de la conséquence, voir la figure 6.

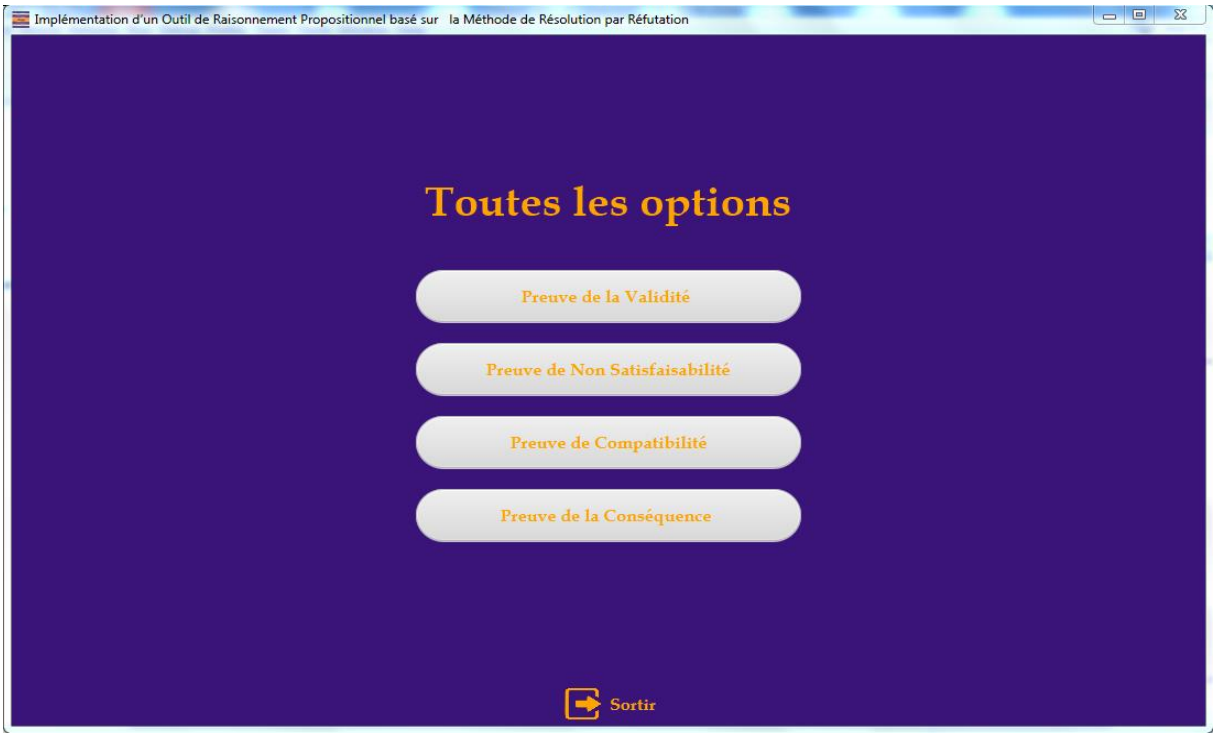


Figure 6: L'interface d'option.

L’interface option nous permet de choisir une des propriétés des formules pour la traiter en utilisant le principe de la méthode de preuve résolution par réfutation.

4.2 Exemples d’utilisation

Nous allons maintenant expliquer chaque propriété logique à travers notre application en traitant quelques exemples et afficher les résultats obtenus.

4.2.1 Preuve de la Tautologie

Lorsque l'utilisateur sélectionne l’option « preuve de la validité », l'interface suivante apparaît. Cette figure sert à écrire la formule propositionnelle qui nous voulons prouver, soit :

$$\varphi = (B \rightarrow (A \rightarrow R)) \rightarrow ((B \rightarrow A) \rightarrow (B \rightarrow R))$$

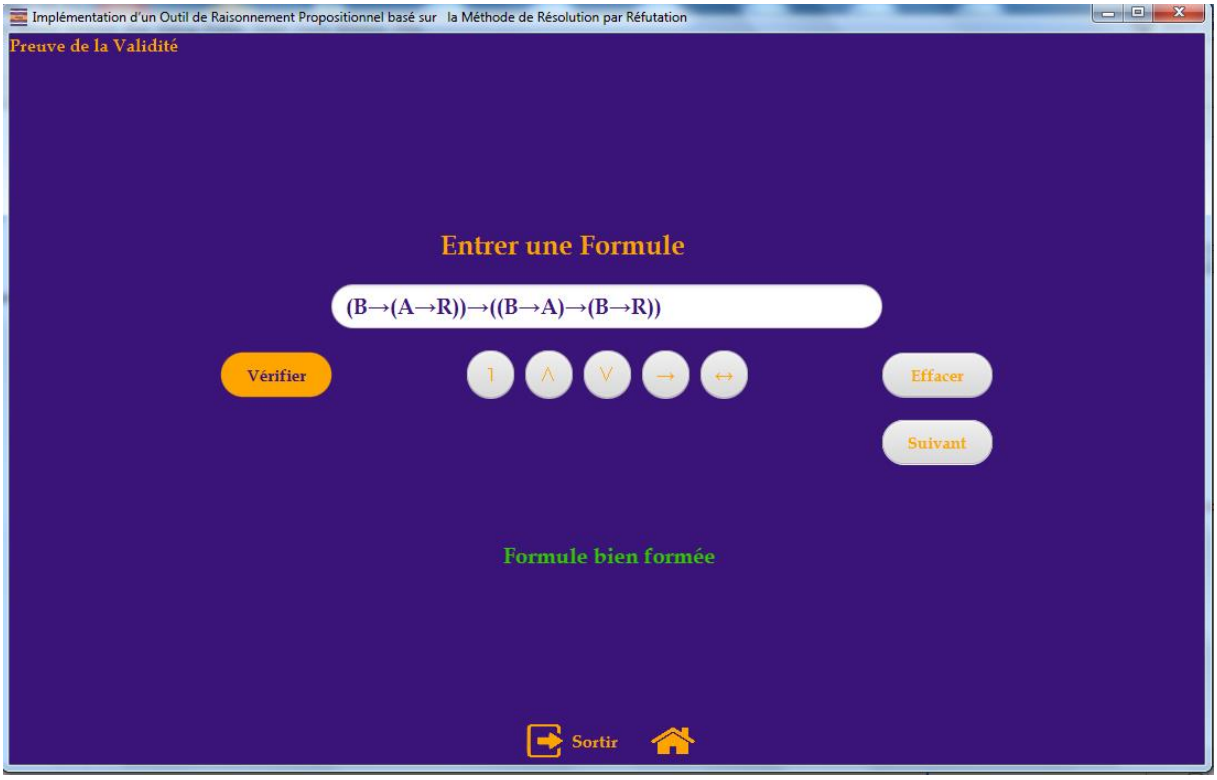


Figure 7: L’interface de preuve de la Validité

Si la formule φ est bien formée (voir chapitre 2), l’outil fournit le bouton « Suivant » pour aller à l’interface de résolution pour le traitement, sinon un message d’erreur sera apparaître afin de vérifier la syntaxe propositionnelle de la formule.

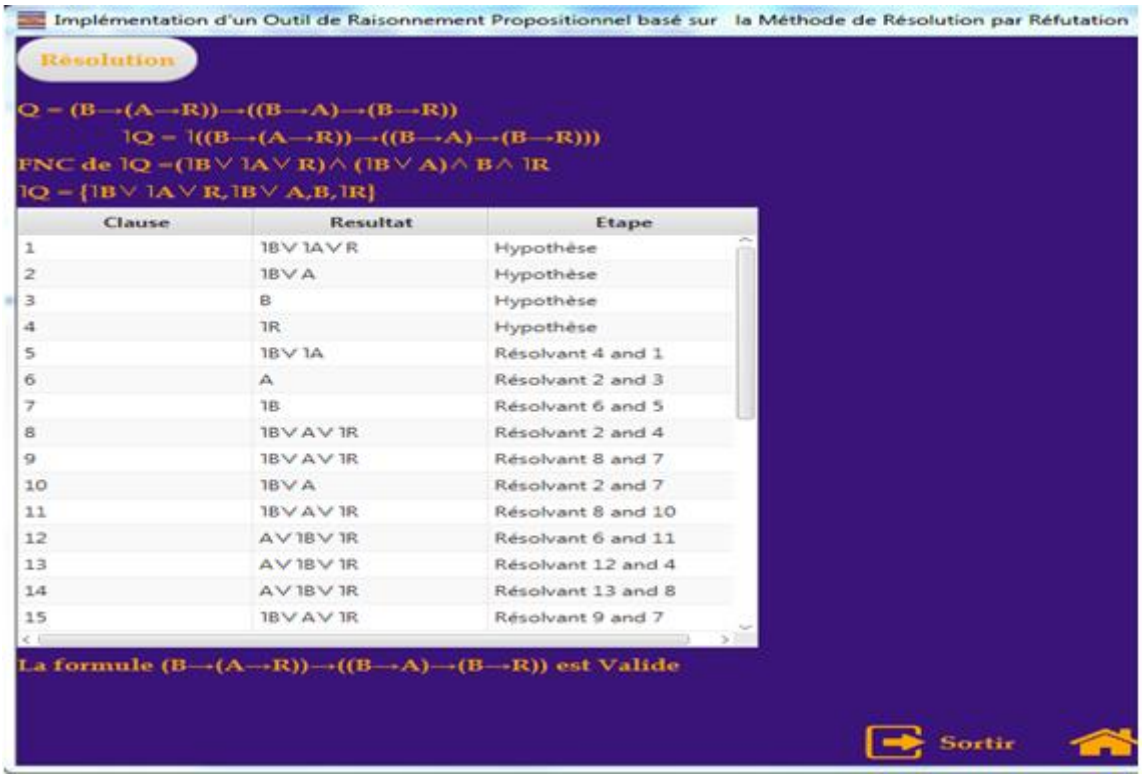


Figure 8 : Traitement de la Validité.

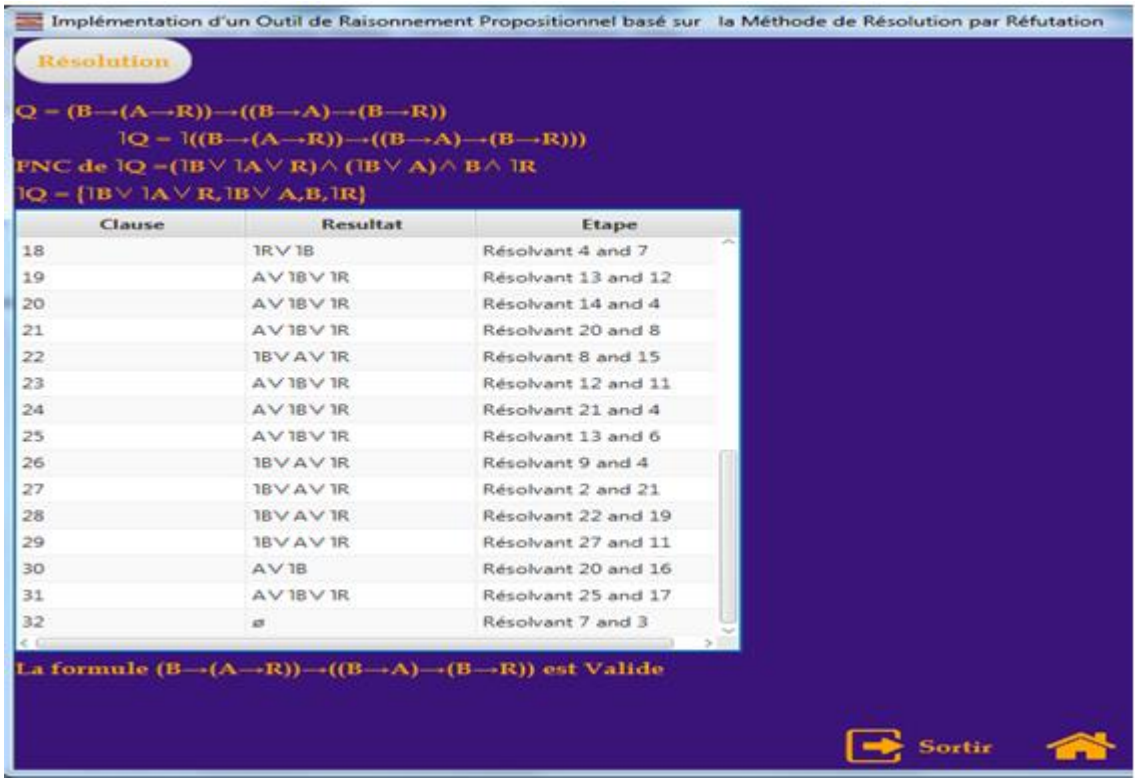


Figure 9: La suite de traitement.

Afin de montrer la tautologie (validité) d'une formule φ en utilisant la méthode de résolution par réfutation, on prouve la déduction de l'ensemble vide à partir de l'ensemble des clauses de sa négation (la formule φ initiale).

Tout d'abord, la négation est ajoutée à la formule φ, donc nous obtenons :

$$\neg\varphi = \neg((B\rightarrow(A\rightarrow R))\rightarrow((B\rightarrow A)\rightarrow(B\rightarrow R))).$$

Puis on calcule le FNC de ¬ φ.

$$\neg\varphi = (\neg B\vee\neg A\vee R)\wedge(\neg B\vee A)\wedge(B\wedge\neg R)$$

Ensuite, les clauses obtenue sont extraites et placées dans un ensemble (voir figure 8).

Si l'ensemble des clauses est groupé, on démarre les calcules. Chaque fois que deux clauses sont sélectionnés où le premier contient un littéral et l'autre contient la négation du même littéral, en d'autres termes les deux clauses contiennent deux littéraux complémentaires. Le système les supprime automatiquement en créant leur résolvant, ce dernier est ajouté dans l'ensemble des clauses initial, L'opération est répété jusqu'à ce que nous obtenons la clause vide si possible, notre but.

A la fin de calcule de la formule φ de l'exemple ci-dessus, nous trouvons la clause vide c'est le résolvant de (clause 3: B et la clause 7: ¬B).
Donc φ est valide.

4.2.2 Preuve de la Non Satisfiabilité

Afin de prouver la propriété du non satisfiabilité, l'utilisateur doit choisir le bouton approprié « Preuve de Non Satisfiabilité », l'interface ci-dessous sera apparaître.

Nous essayons d'expliquer le principe à travers un exemple, soit la formule suivant :

$$\varphi = \neg((a\rightarrow b)\wedge(a\vee c)\rightarrow b\vee c)$$

Cette partie concerne l'insatisfaction, où l'utilisateur entre une formule après laquelle la syntaxe est vérifiée, comme la montre la figure ci-dessous. Si la formule est bien formée, le bouton suivant apparaît, qui permet d'accéder à l'interface de traitement de l'insatisfaction.

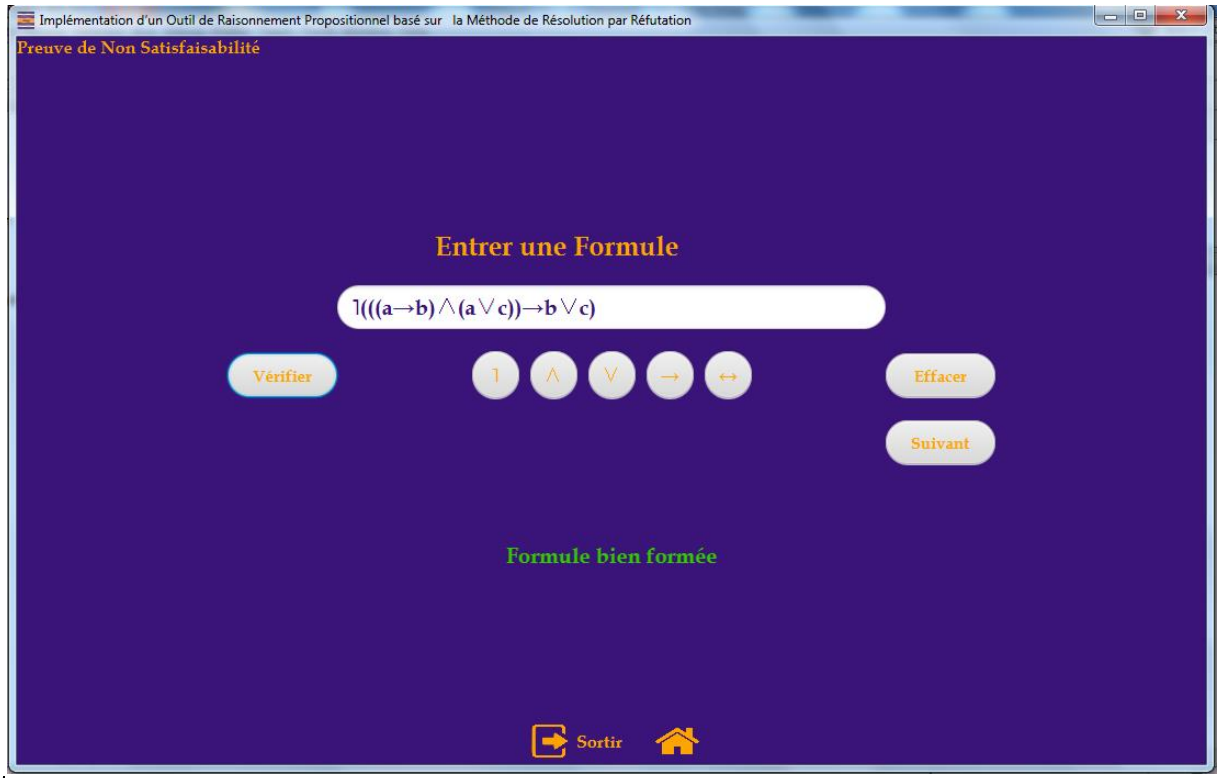


Figure 10 : L'interface de preuve de Non Satisfiabilité et vérification d'une formule.

Quand en cliquant sur bouton résolution nous avons voir la fenêtre suivante :

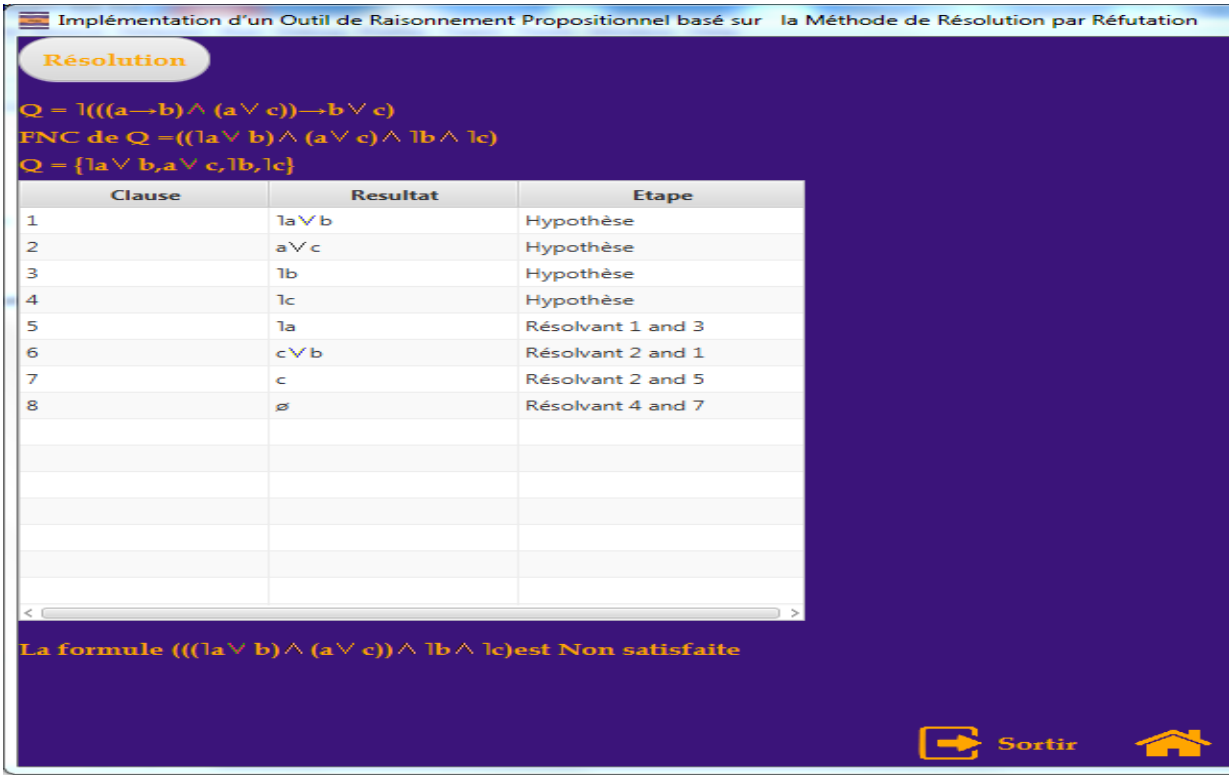


Figure 11 : Traitement de preuve de Non Satisfiabilité

Afin de montrer l'antilogie (non satisfiabilité) d'une formule φ en utilisant la méthode de résolution, on extraire directement la forme normale conjonctive.

$$\varphi\ (FNC) = (\neg a \vee b) \wedge (a \vee c) \wedge \neg b \wedge \neg c$$

Puis on démarre le calcule par le même principe jusqu'à attendre la clause vide si possible bien sur. Dans cet exemple on obtient la clause vide par les deux résolvants, quatre : ¬c et le résolvant 7 : c, donc φ est non satisfaite.

4.2.3 Preuve de la Compatibilité d'un ensemble de formules

Pour prouver la compatibilité d'un ensemble de formules, notre application fournit une option d'ajout qui permet de saisir le nombre de formules voulu. L'interface ci-dessous montre cette astuce, noté que chaque formule doit être vérifiée syntaxiquement en cliquant sur le bouton vérifier.

L'exemple suivant traite la compatibilité des trois formules :

$\varphi 0 = a \rightarrow c$

$\varphi 1 = a \wedge \neg c$

$\varphi 2 = a \vee b$

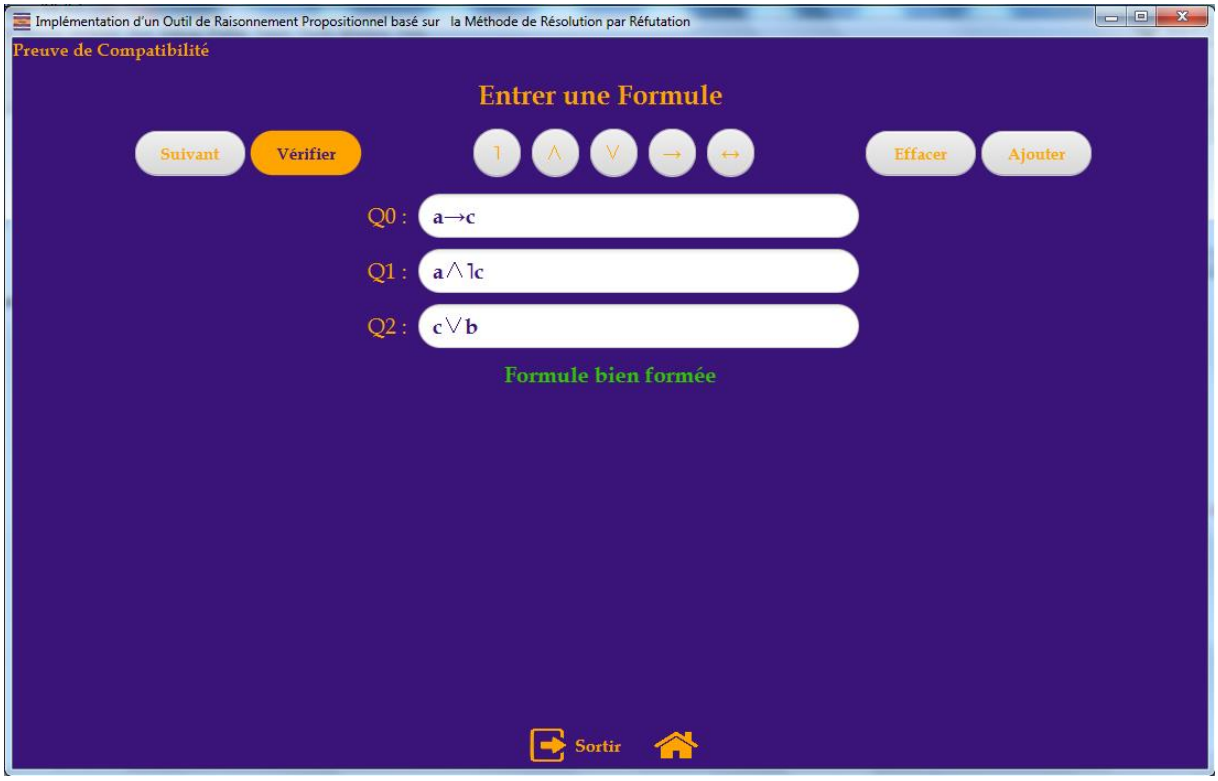


Figure 12: L'interface de preuve de la Compatibilité

Après la vérification syntaxique des formules, l'outil fournit la fenêtre de résolution (voir la figure ci-dessous).

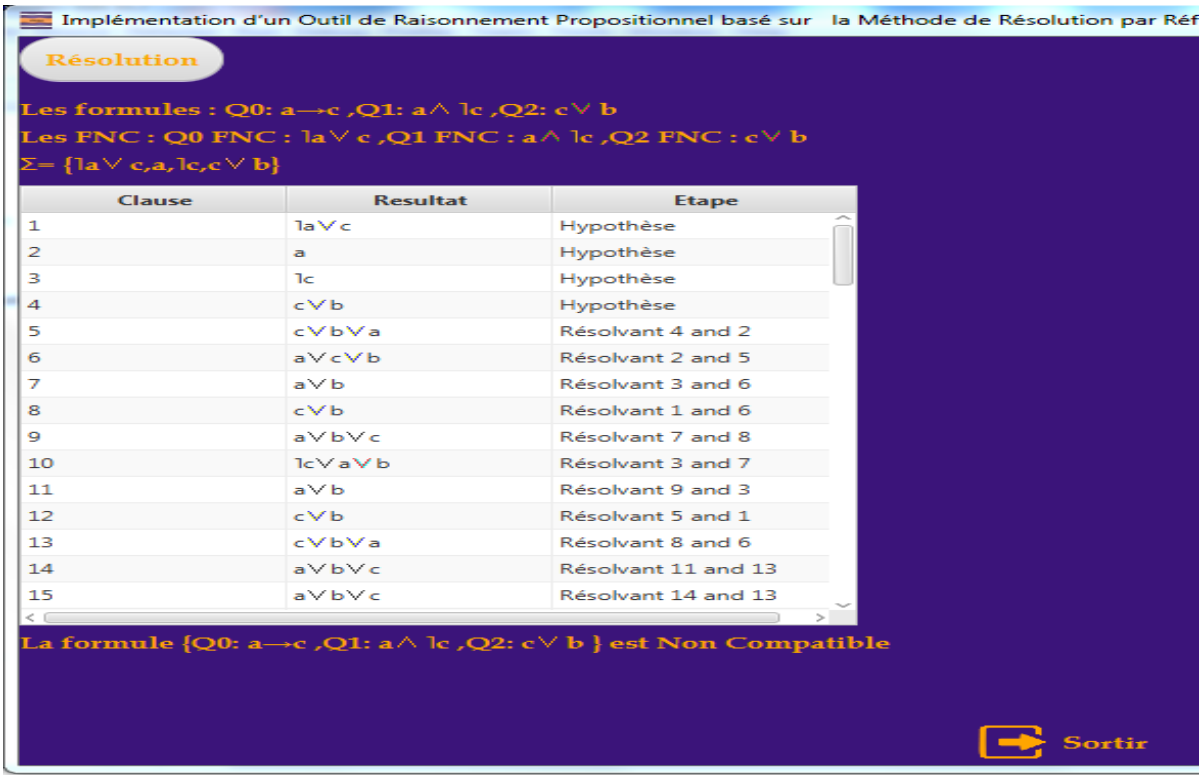


Figure 13 : L'interface de traitement de preuve de la Compatibilité.

Après le traitement de vérification syntaxique de l'ensemble des formules traitées. Le système cherche le FNC de chaque formule saisie (dans notre cas de l'exemple, seulement trois formules), puis le système regroupe l'ensemble des clauses fournis par chaque FNC. Enfin, le calcul doit être démarré en appliquant la méthode de résolution. Chaque fois que deux clauses contiennent des littéraux complémentaires, le système les supprime et ajoute leur résolvant dans l'ensemble initial des clauses. Ce processus est répété jusqu'à avoir l'ensemble vide si possible.

Si on trouve que le résultat était un ensemble vide, alors l'ensemble traité est incompatible, sinon l'ensemble des formules est compatible. L'interface 13 montre l'incompatibilité des trois formules φ0, φ1 et φ2.

4.2.4 Preuve de la conséquence

Afin d'expliquer la prouver de la conséquence logique en utilisant notre application, nous donnons un exemple :

$$\neg a \rightarrow c,\ a \wedge \neg b,\ b \leftrightarrow c,\ \models a$$

Cette partie de notre application sert à traiter la conséquence non seulement entre deux formules mais la conséquence d'une formule par un ensemble des formules (voir l'exemple).

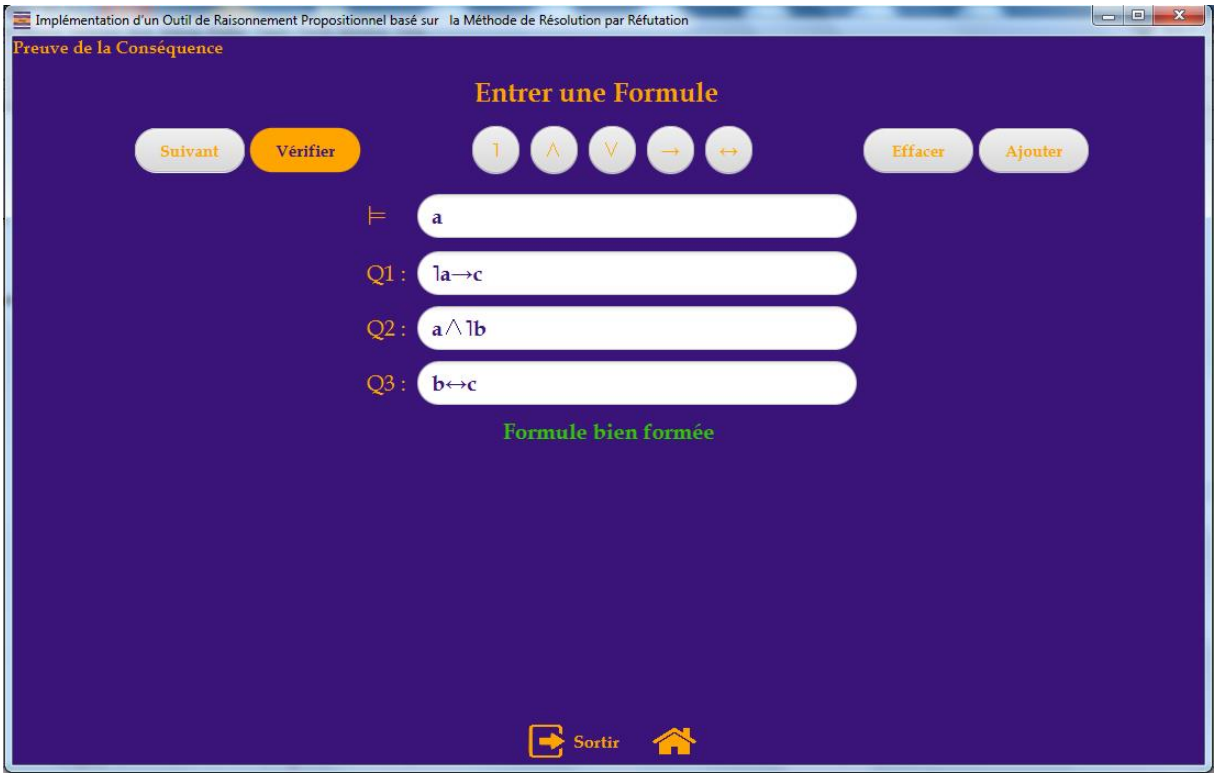


Figure 14: L'interface de preuve de la conséquence

Si toutes les formules entrées sont bien formées, l'outil fournit un bouton « Suivant » pour aller à l'interface de résolution pour le traitement de la conséquence.

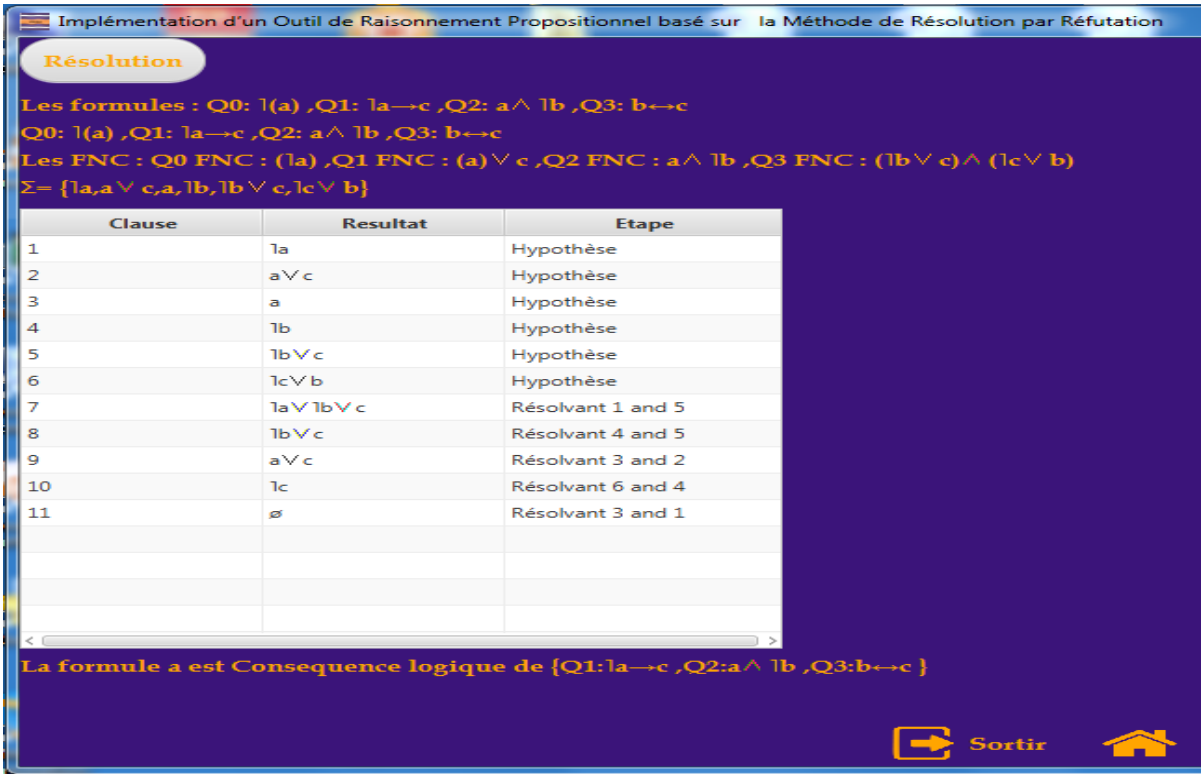


Figure 15 : Traitement de preuve de la conséquence.

Prouver que : ¬a→c, a ∧¬b, b↔c, ⊨ a

Revient à prouver que la FP : (¬a→c) ∧ (a ∧¬b) ∧ (b↔c) → a est tautologie.

Nous avons vu dans la section 4.2.1 (preuve de la validité) que pour prouver la tautologie d'une formule propositionnelle il faut prouver l'insatisfiabilité de sa négation.

Donc : ¬((¬a→c) ∧ (a ∧¬b) ∧ (b↔c) → a) = (¬a→c) ∧ (a ∧¬b) ∧ (b↔c) ∧ ⌉a
= (¬a→c), (a ∧¬b), (b↔c), ⌉a ⊨ ∅

Alors, la première étape était de trouver le FNC de chaque formule. La deuxième de regrouper l'ensemble des clauses et la troisième d'appliquer le calcul de résolution jusqu'à ce que le système obtient la clause vide pour qu'il soit arrêter. Dans ce cas la clause vide est le résolvant de (clause 1: ¬A et clause 3 : A).

Donc A est conséquence logique de (A ∧¬B, B↔C, ¬A→D)

5 Conclusion

Dans ce dernier chapitre, nous avons évoqué les objectifs de conception de ce projet, après nous avons présenté l'environnement matériel et logiciel ainsi que les langages utilisés pour construire cette application. Puis, nous avons montré les différentes interfaces à travers des captures d'écran et mentionné les éléments de chaque interface avec une explication détaillée de leur fonctionnement à l'aide d'un ensemble des exemples.

Conclusion Générale

Le calcul propositionnel est un système logique dans lequel on se donne un ensemble de variables propositionnelles qui peuvent être vraies ou fausses, permettant de discuter des connecteurs logiques et de parler des fonctions booléennes.

Le calcul propositionnel tient une grande place en informatique, puisque nos ordinateurs actuels sont digitaux, et travaillent en binaire. Pour définir formellement et proprement ce langage, nous devons distinguer la syntaxe de la sémantique : la syntaxe décrit comment on écrit les formules. La sémantique décrit leur sens. Nous pouvons classer les formules propositionnelles selon leur interprétation à des formules satisfaites non valides, tautologies, antilogies. Pour prouver l'appartenance d'une formule à une telle classe, ou pour prouver la compatibilité d'un ensemble des formules ou la conséquence, il soit possible d'énumérer toutes les évaluations afin d'obtenir une table de vérité. Cette méthode est coûteuse en espace et en temps.

L'objectif de notre projet de fin d'étude était d'implémenter et développer une application de raisonnement propositionnel basé sur la méthode de résolution par réfutation. Cette méthode est l'une des méthodes de systèmes de preuve qui démontrent toutes les propriétés logiques des formules.

Le point de départ de la réalisation de ce projet était une récolte des informations nécessaires sur la logique propositionnelle et leur système de preuve en particulier la méthode de résolution.

Notre projet sert à automatiser la démonstration des différentes propriétés de formules telles que la satisfaction, la validité, la compatibilité et la conséquence soit entre deux formules ou avec un ensemble de formules.

Egalement, nous avons montré l'efficacité et la simplicité d'utilisation de notre application, à travers quelques interfaces significatives dans la dernière partie de ce manuscrit.

L'apport de ce travail a été une importance très considérable, en effet, il nous a permis de suivre une méthodologie de travail bien étudiée et d'approfondir nos connaissances dans le monde de développement des applications.

Cette expérience nous a permis de maîtriser le langage de développement java (l'IDE Netbeans), sous lequel, le développement n'a pas été une tâche facile, mais nous n'avons pas hésité à y participer. Ce projet a était un déclencheur pour commencer à s'intéresser à ce domaine. Nous ne comptons pas nous arrêter ici, mais continuer à développer nos compétences et plonger encore dans ce domaine.

Références Bibliographiques

[1] : Dr ADEL née AISSANOU Karima, Support de cours Logique Mathématique, Université A/Mira de Béjaia, 2015-2016.

[2] : Jean-Louis Rouget, Logique, Université cote D'Azur, 2007.

[3] : BENKADDOUR Fatima Zohra, introduction a la logique mathématique, Département des sciences exactes, ENS d'Oran, 2018-2019.

[4] : Jean-Michel Richer, logique et calcul propositionnelle, Université d'Angers, 2008.

[5] : Pascal Amsili, Logique du premier ordre, Université de Paris 7, 2006.

[6] : Ralf Treinen, Outils Logiques, Université Paris Diderot, Paris 7, 2012.

[7] : Antoine Noël De Tilly, Le raisonnement à base de logique propositionnelle à l'appui de la fusion et de la révision de bases de données géospatiales, Mémoire de Maitre des Sciences (M.Sc.), Université Laval, Québec, 2007.

[8] : N. Hameurlain, Logique des programmes, Université de Pau et des Pays de l'Adour, 2014.

[9] : Brice Halimi, Introduction à la logique: logique propositionnelle, Université Paris Nanterre.

[10] : P. Gribomont, Logiques pour l'intelligence artificielle, Montefiore Institute, Liège Université, 2006.

[11] : Laurent Audibert, Article: logique du raisonnement valide, 2007, https://laurent-audibert.developpez.com/Cours-Logique/

[12] : Pascal Lafourcade, Stéphane Devismes, Michel Lévy, Logique et démonstration automatique: Une introduction à la logique propositionnelle et à la logique du premier ordre, 2018.

[13] : Dictionnaire de mathématique, http://www.bibmath.net/dico/index.php?action=affiche&quoi=./f/formenormale.html, 23-03-2021

[14] : Séverine Fratani et Luigi Santocanale, Cours Logique et Calculabilité, Université Aix-Marseille, 2017.

[15] : Christine Paulin-Mohring, Eléments de logique pour l'informatique, https://www.lri.fr/~paulin/Logique/html/cours004.html , 20-3-2021.

[16] : Zeghib Nadia, Logique propositionnelle, université Constantine 2, 2015

[17] : Andreas Herzig, Introduction à la logique, 2016, Université Paul Sabatier, https://www.irit.fr/~Andreas.Herzig/C/prop.html#sema, 20-3-2021.

[18] Jacques Bergeron, Utilisation des tableaux sémantiques dans les logiques de description, Université de Montréal, mars 2011.

[19] Francois Schmitz, Logique propositionnelle, Université de Nautes, 2014.

[20] Audibert Laurent, Logiques du raisonnement valide, 2004.

[21] Peter Habermehl, Logique : Les systèmes de preuves syntaxiques, Université Paris Diderot, 2015.

[22] Marie-Pierre Gleizes, Principe de résolution en logique des propositions, Université Claude Bernard Lyon 1, 2002.

[23] Christian Retoré, Calcul Propositionnel, Université de Nantes, 2001.

[24] https://studylibfr.com/doc/528243/7-logique-propositionnelle---iii-la-m%C3%A9thode-de-r%C3%A9solution

[25] Mr HABET Md-Said, Logique Mathématique, Université Mouloud MAMMERI de Tizi-Ouzou, 2019/2020.

[26] https://www.techno-science.net/glossaire-definition/NetBeans.html.

[27] https://www.futura-sciences.com/tech/definitions/internet-java-485.

[28] www.techopedia.com/definition/5594/java-development-kit-jdk/.

[29] https://mikarber.developpez.com/tutoriels/java/introduction-javafx/.