République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de Recherche Scientifique
Université Kasdi Merbah-Ouargla
Faculté des Nouvelles Technologie de l'information et de la Communication
Département d'Informatique et des Technologies de l'Information

Mémoire fin d'études
pour l'obtention du diplôme de Master en Informatique Fondamentale

Spécialité : Informatique Fondamentale

Thème :

Environnement propositionnel basé systèmes de preuve

Réalisé par :

- HATHAT Mohamed
- MILOUDI Wiaam

Devant le jury composé de :

• Dr.SAADI Wafa	UKM-Ouargla	Président du jury
• Dr.CHAMA Wafa	UKM-Ouargla	Superviseur
• Dr.ZITOUNI Farouk	UKM-Ouargla	Membre du jury

Soutenue le : 19/06/2022

Année Universitaire : 2021/2022

Dédicaces

Je rends grâce à ALLAH le TOUT PUISSANT pour tous les bienfaits dont il comblé Ce mémoire ayant été rédigé. je le dédie : En premier lieu à ma mère et à mon père qui ont consenti beaucoup de sacrifices pour me permettre de réaliser mes objectifs qu'ils trouvent ici toute ma reconnaissance et ma gratitude. Particulièrement à mes chères sœur : AEML, AMINA, ASMA et INASE, ainsi qu'à mes chers frères : AHMED, Yousef et ISMAIL. Et a mon binôme Miloudi Wiaam A toute ma famille surtout ma grande et tous mes proches mère A tous mes camarades et mes amis surtout Mahdi, Abdel Moneim, Marwan, Al-Muntasir Billah. A tous mes enseignants depuis le primaire jusqu'à cet instant.

HATHAT Mohamed

Je rends grâce à ALLAH le TOUT PUISSANT pour tous les bienfaits dont il comblé Ce mémoire ayant été rédigé. je le dédie : En premièrement ma mère et à mon père qui ont consenti beaucoup de sacrifices pour me permettre de réaliser mes objectifs qu'ils trouvent ici toute ma reconnaissance et ma gratitude. Particulièrement à mon mari et mes chères sœur, ainsi qu'à mes chers frères. Et a mon binôme HATHAT Mohamed A toute ma famille A tous mes camarades et mes amis. A tous mes enseignants depuis le primaire jusqu'à cet instant.

MOLOUDI Wiaam

Remerciments

Je remercie Allah, le tout puissant, le miséricordieux, de avoir appris ce que j'ignorais, de avoir donné la santé et la patience et tout ce dont avais besoin pour réaliser le travail imposé et rédiger ce mémoire. Je tiens également à exprimer mes vifs remerciements chaleureusement à notre encadreur , ***Dr.Chama Wafa*** *pour avoir abord proposé ce thème, pour le suivi continu tout au long de la réalisation de ce travail et qui a pas cessé de me donner ses conseils. Sans oublier les membres de jury :Dr.SAADI Wafa et Dr.ZITOUNI Farouk Nos remerciements vont aussi au département d'informatique et des technologies de l'information ,et Nous remercions tous les enseignants qui ont transmis leur message scientifique et moral avec honnêteté*

Résumé

L'utilisation de l'outil informatique dans tous les domaines joue un rôle très important surtout pour ceux qui se basent sur l'exactitude. Ce projet représente la mise en œuvre d'un logiciel d'analyse sémantique pour les formules propositionnelles. Ce projet de fin d'étude propose quelles que techniques idéales afin d'étudier les propriétés des formules logiques, où un outil a été développé basé sur trois méthodes de preuve, la méthode de résolution par réfutation, la méthode des tableaux sémantiques et le diagramme de Quine.

Notre objectif d'une part est de détailler chaque méthode en l'implémentant afin de faciliter leur mise en œuvre et d'autre part de faciliter leur utilisation en proposant des techniques d'optimisation.

Mots clés : logique propositionnelle, formule, satisfiabilité, validité, compatibilité, conséquence, diagramme de Quine, tableau sémantique, résolution, réfutation.

Abstract

The use of computers in all areas is a very important role, especially for those which base on accuracy. This project represents the implementation of semantic analysis software for propositional formulas. This end-of-study project proposes some ideal approaches to study the properties of logical formulas, where a tool has been developed based on three proof methods, resolution by refutation method, semantic tables method and Quine's diagram.

Our objective, on the one hand is to detail each method by implementing a tool that serves to facilitate their use and on the other hand to facilitate their use by proposing optimization techniques.

Keywords : propositional logic, formula, satisfiability, validity, compatibility, consequence, Quine's diagram, semantic table, resolution, refutation.

الملخص

يلعب استخدام الكمبيوتر في جميع المجالات دورا مهما للغاية، خاصة لأولئك الذين يعتمدون على الدقة. يمثل هذا المشروع تنفيذ برنامج تحليل دلالي للصيغ المنطقية. يقترح مشروع نهاية الدراسة بعض التقنيات المثالية لدراسة خصائص الصيغ المنطقية، حيث تم تطوير أداة بناءًا على ثلاث طرق للإثبات، طريقة الحل عن طريق التنفيذ، وطريقة الجداول الدلالية ومخطط كوين. هدفنا من ناحية هو تفصيل كل طريقة من خلال برمجتها من أجل تسهيل تنفيذها ومن ناحية أخرى تسهيل استخدامها من خلال اقتراح تقنيات التحسين.

الكلمات المفتاحية المنطق الإفتراضي، الصيغة، الإرضاء، الصلاحية، التوافق، النتيجة المنطقية، مخطط كوين، الجدول الدلالي ، القرار، التفنيد.

Table des Matiéres

Liste des Figures

Liste des Tableaux

Introduction Générale

La logique mathématique est née à la fin du 19 e siècle, à cette époque il y avait beaucoup de problèmes en mathématiques. Les fondateurs de la logique ont proposé les concepts et les bases essentiels afin de résoudre la crise des fondements provoquée par la complexité des mathématiques, démonstration des énoncés non encore prouvés et l'apparition des paradoxes. Cela, en représentant les énoncés mathématiques sous forme de formules ensuite les démontrer avec des méthodes de raisonnement rigoureuses.

On peut définir la logique comme « la science qui étudie les règles générales du raisonnement correct ». Donc, le but d'étudier la logique est :

— Raisonner correctement
— Résoudre les problèmes complexes
— Trouver les solutions rapidement

L'utilisation de la logique mathématique n'est pas limitée dans le domaine théorique pur, mais elle a fortement contribué à la naissance des premiers ordinateurs grâce à la binarité des valeurs de vérité. Ces dernières sont la base de tous les circuits électronique qui composent l'ordinateur. La logique est utilisée en informatique pour modéliser de manière formelle des "objets" rencontrés par les informaticiens, ces concepts ont beaucoup contribué dans les applications de l'intelligence artificielle, les bases de données, bases de connaissances, pré-post conditions d'une procédure, etc. L'informaticien doit être capable de se servir du modèle et raisonner sur celui-ci, comme la validation d'un modèle de données, prise de décision à partir des faits et d'une base de connaissances, preuve de correction d'une procédure/ d'un programme.

La logique a plusieurs types, nous somme concentrés au calcul des propositions qu'est la base de toute définition logique et même de tout raisonnement. Le calcul des propositions est parfois appelé logique des propositions, logique propositionnelle, calcul des énoncés, théorie des fonctions de vérité et aussi logique booléenne.

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. Afin de surmonter ce problème, il est préférable d'utiliser les 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, nous avons travaillé sur le diagramme de Quine, la

méthode des tableaux sémantiques et la méthode de résolution par réfutation.

Cependant, leurs utilisation manuelle est très difficile et peut provoquer des erreurs. Nous avons développé un outil qui englobe ces trois méthodes de preuve afin de faciliter la tâche aux étudiants, chercheurs, mathématiciens, informaticiens et de tout utilisateur par la diminution du temps de calcule et garanti les résultats obtenus.

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. Également, nous avons présenté la méthode de Quine en essayant de la clarifier par des exemples détaillés. Le deuxième chapitre propose un ensemble de méthodes, nous avons mis l'accent sur la méthode des tableaux qui aide à déterminer la valeur de vérité des formules non simples, plus compliquées, et surtout demandent beaucoup de temps et d'efforts. Enfin, le dernier chapitre présente une méthode fameuse, la méthode de résolution par réfutation en détail.

Chapitre 1

Notions de base et diagramme de Quine

1.1 Introduction

L'objet de la logique propositionnelle est l'étude des propositions et de certaines opérations qui permettent de les combiner. Pour étudier la syntaxe d'un langage il faut donner un alphabet (un ensemble de symboles) et des règles de constructions syntaxiques d'expressions à partir de ces symboles. Nous montrons dans ce chapitre d'où proviennent ces objets et ce qu'ils sont.

1.2 Formule bien formée

1.2.1 Le langage propositionnel

Le langage propositionnel est composé de formules représentant des propositions. Comme les autres langages, le langage du calcul propositionnel est caractérisé par sa syntaxe et sa sémantique.

1.2.2 L'alphabet

L'alphabet est composé des symboles du langage. Il comporte :

- Un ensemble dénombrable de variables propositionnelles.
  généralement, on utilise les lettres de l'alphabet latin (a, b, c ... et z ou A,B,C ...et Z) éventuellement indicées.
  Exemple : A, A123 , c98

- De deux constantes : vrai et faux

- D'un ensemble de connecteurs logiques : conjonction (notée $\wedge$), disjonction (notée $\vee$), implication (notée $\rightarrow$), équivalence (notée $\leftrightarrow$), négation (notée $\neg$)

- Symboles impropres : les séparateurs (ou parenthèses) "(" et ")", qui sont importantes car elles délimitent la fin des sous formules et parfois la priorité des connecteurs.

1.2.3 Les connecteurs logiques

Négation (notée $\neg$)

Si "p" désigne une proposition alors on note souvent "$\neg$ p " comme la négation de "p" qui est elle-même une proposition. "$\neg$p " signifie : il n'est vraie que "p" [12].

Exemple : "Il n'est pas vraie que 8 est un nombre premier "

Conjonction et Disjonction (et notée $\wedge$) et (ou notée$\wedge$)

Si "p" et"q" désignent deux propositions, alors on note souvent :

"p $\wedge$ q" une proposition, pour exprimer le fait que les deux assertions p et q sont toutes les deux sont vraies.[12]

"p $\vee$ q" une proposition, pour exprimer le fait que au moins l'une des deux assertions p et q est vraie (elles peuvent être vraies toutes les deux).[12]
Exemple : L'entier 5 est impair et l'entier 2 est pair, ou tout triangle a trois cotés. Nous symbolisons chaque phrase par une lettre :
L'entier 5 est impair par I.
L'entier 2 est pair par P.
tout triangle a trois cotés par T.
la modélisation est :(I $\wedge$ P) $\vee$ T

Implication (notée $\rightarrow$)

Très importante pour exprimer des informations conditionnelles et contextuelles. Si "p" et "q" désignent deux propositions, alors on note souvent :
"p$\rightarrow$q", pour exprimer le fait que si p est vraie alors q également vraie.
Noter que, si la proposition "p" est fausse alors l'implication "p$\rightarrow$q" est considérée comme vraie.L'implication matérielle "p$\rightarrow$q" est logiquement équivalente à "$\neg$ p$\vee$q" [12]

Exemple : s'il pleut alors le ciel est nuageux. Nous symbolisons chaque phrase par une lettre :
il pleut par P
le ciel est nuageux N
formule est : P $\rightarrow$N
C'est une proposition vraie

Equivalence (notée $\leftrightarrow$)

Si "p" et "q" désignent deux propositions alors on note souvent :
"p $\leftrightarrow$ q", pour exprimer le fait que les deux propositions p et q sont équivalentes. "p $\leftrightarrow$ q" est vue comme une conjonction de "p$\rightarrow$q" et "q$\rightarrow$p". [12]

Exemple : Omar va au stade si et seulement si le jour est Vendredi.

Nous symbolisons chaque phrase par une lettre :
Omar va au stade par S
le jour est Vendredi par V
formule est :S $\leftrightarrow$ V

1.2.4 Reconnaissance des formules bien formées

Afin de bien reconnaitre les formules par notre outil, nous avons proposé d'utiliser la notion d'automate. Les automates qui sont des machines qui, après avoir exécuté un certain nombre d'opérations sur le mot, peuvent répondre si le mot est une formule ou non. Les automates sont dits les reconnaisseurs de langages.

Automate

Un automate à états finis (AEF) est une machine abstraite définie par le quintuplés A=(Σ ,Q, e0, F, δ) tel que :[10]

- Σ est l'ensemble des symboles formant les mots en entrée (l'alphabet Σ des mots à reconnaître)
- Q est l'ensemble des états possibles de A
- e0 est l'état initial
- F est l'ensemble des états finaux (F $\neq \emptyset$, F $\subseteq$ Q). F représente l'ensemble des états d'acceptation
- $\delta : Q \times \Sigma \rightarrow Q$ est la fonction de transition de A. δ est définie de Q $\times\Sigma$ dans Q. Elle permet de passer d'un état à un autre selon l'entrée en cours :
  $\delta(e_i$,a)= $\{e_1, ..., e_n\}$ ou $\emptyset$ ($\emptyset$ signifie que la configuration n'est pas prise en charge)

Il existe trois principales représentations pour les AEF, à savoir :

1. Sous forme de fonction (relation).
2. Représentation matricielle (sous forme de table)
3. Représentation graphique (sous forme de graphe orienté étiqueté)

Les automates sont souvent donnés sous la forme d'un graphe : les états sont les nœuds du graphe et les arcs correspondent à la fonction de transition.

Exemple :

Σ={a,b}
Q={e0,e1,e2,e3}
e0={e0}
F={e3}

δ(e0,a)= {$e1$}/ δ(e0,b)= {$e0$}/ δ(e1,b)= {$e2$}/ δ(e2,b)= {$e3$}

Représentation graphique : Pour faciliter la tâche de la programmation, nous

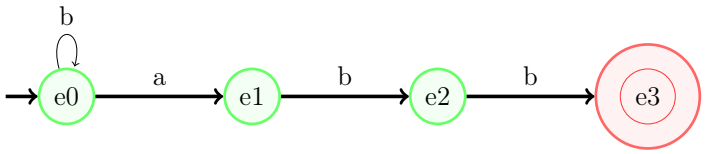


FIGURE 1.1 – Représentation graphique d'un automate

avons utilisé la représentation matricielle (sous forme de table).
La table possède autant de lignes qu'il y a d'états dans l'automate de telle sorte que chaque ligne correspond à un état. Les colonnes correspondent aux différents symboles de l'alphabet (voir l'exemple ci-dessous).
Exemple :(suite)

	a	b
e0	**e1**	**e0**
e1	/	**e2**
e2	/	**e3**
e3	/	/

TABLE 1.1 – La matrice de transition pour l'automate 1.1 :

1.2.5 Notre automate proposé

Notre automate à états finis proposé accepte toute formule bien formée. Une formule doit être écrite sous l'une des formes suivantes :

- Atome ou la négation d'atome (littéral) : A1, b, ¬A,...
- Combinaisons des littéraux (par des connecteurs binaires) tels que :
  — Chaque formule doit être saisie entre parenthèses
  — Le nombre de parenthèses ouvrantes égal au nombre des parenthèses fermantes.

  Exemple :
  $\varphi = (\varphi 1 \wedge \varphi 2)$
  $\varphi = ((\varphi 1 \vee \varphi 2) \rightarrow \varphi 3)$

Représentation relationnelle

Σ={a-zA-Z,0-9, $\vee, \wedge, \rightarrow, \leftrightarrow$,$\neg$,(,)}
Q={e0,e1,e2,e3,e4,e5,e6}
q0={e0}
F={e1,e4}
δ(e0,$\neg$, ()= {$e1$} / δ(e0,b)= {$e0$}
δ(e1,0-9)= {$e1$} / δ(e1,$\vee\wedge \rightarrow\leftrightarrow$)= {$e2$} / δ(e1,))= {$e4$}
δ(e2,$\neg$)= {$e2$} / δ(e2,a-zA-Z)= {$e3$} / δ(e2,()= {$e3$}
δ(e3,0-9)= {$e3$} / δ(e3,))= {$e4$}
δ(e4,))= {$e4$} / δ(e4,$\vee\wedge \rightarrow\leftrightarrow$)= {$e2$}
δ(e5,$\neg$, ()= {$e5$} / δ(e5,a-zA-Z)= {$e6$}
δ(e6,0-9)= {$e6$} / δ(e6,$\vee\wedge \rightarrow\leftrightarrow$)= {$e2$} / δ(e6,))= {$e4$}

Représentation graphique

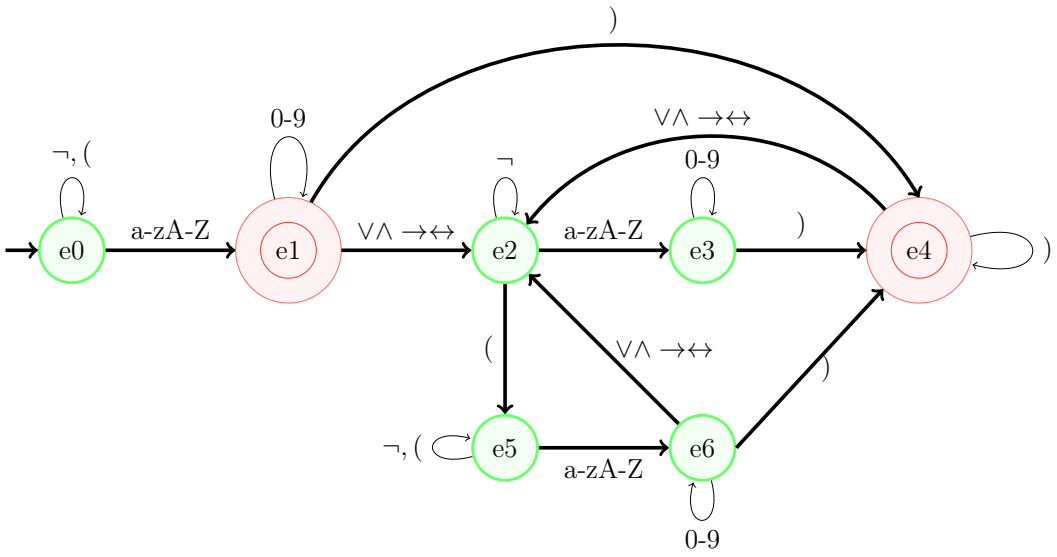


FIGURE 1.2 – Automate proposé pour la reconnaissance des FBF

Représentation Matricielle :

	a-zA-Z	0-9	$\vee\wedge\rightarrow\leftrightarrow$	$\neg$	(	)
e0	**e1**	/	/	**e0**	**e0**	/
e1	/	**e1**	**e2**	/	/	**e4**
e2	**e3**	/	/	**e2**	**e5**	/
e3	/	**e3**	/	/	/	**e4**
e4	/	/	**e2**	/	/	**e4**
e5	**e6**	/	/	**e5**	**e5**	/
e6	/	**e6**	**e2**	/	/	**e4**

TABLE 1.2 – Reconnaissance des FBF par la représentation matricielle

Exemple : $\varphi 1 = (A1 \wedge B)$
$\varphi 2 = ((A \vee (B \wedge C)) \rightarrow d\ )$

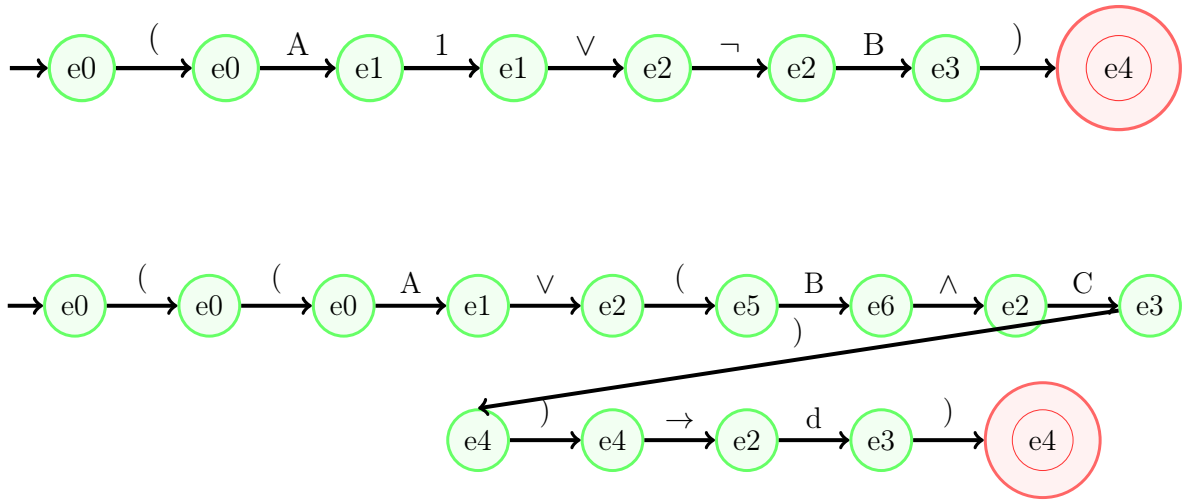


FIGURE 1.3 – Exemple d'exécution des FBF par notre automate proposé

1.3 Sémantique propositionnelle

L'étude de 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.
La sémantique associe une fonction de valuation V pour chaque variable propositionnelle notée :

$$\mathrm{V} : \mathrm{VP} \rightarrow \{1, 0\}\,;$$

où VP est l'ensemble des variables propositionnelles,1 signifie vrai et 0 signifie faux[7].

1.3.1 La table de vérité

La sémantique associe une fonction de valuation qu'est décrite par un graphe appelé table de vérité (ou tableau de vérité).
A chaque formule α à n variables propositionnelles correspond une fonction de vérité unique. Le graphe de cette fonction est défini par une table de vérité à 2^n lignes représentant la valeur de vérité de α correspondant à chaque combinaison de valeur de vérité des n variables (appelées aussi distribution de valeurs de vérité des variables)[7].

1.3.2 Sémantique des connecteurs logiques

La table de vérité de la négation

Si la proposition p est vraie alors ¬ p est fausse.

p	$\neg p$
0	**1**
1	**0**

TABLE 1.3 – Sémantique de la négation

La table de vérité de la conjonction

La conjonction de deux propositions p et q est vraie si et seulement si les deux propositions p et q sont vraies simultanément. D'où la table de vérité suivante :

p	q	$p \wedge q$
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

TABLE 1.4 – Sémantique de la conjonction

La table de vérité de la disjonction

La disjonction de deux propositions est fausse si et seulement si les deux propositions p et q sont fausses simultanément. D'où la table de vérité suivante :

p	q	$p \vee q$
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

TABLE 1.5 – Sémantique de la disjonction

La table de vérité de l'implication

L'implication de deux propositions est fausse dans le cas où p est vraie et q est fausse. Elle est définie par le tableau suivant :

p	q	$p \rightarrow q$
0	**0**	**1**
0	**1**	**1**
1	**0**	**0**
1	**1**	**1**

TABLE 1.6 – Sémantique de l'implication

La table de vérité de l'équivalence

L'équivalence de deux propositions est vraie dans le cas où p et q ont la même valeur de vérité. Elle est définie par le tableau suivant :

p	q	$p \leftrightarrow q$
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

TABLE 1.7 – Sémantique de l'équivalence

1.3.3 Les propriétés logiques

Satisfiabilité

Une formule est dite satisfiable si et seulement si sa table de vérité contient au moins une ligne où la valeur de vérité de α est vraie $[\alpha]_v = 1$.
α est dite insatisfaisable si elle est fausse sur toutes les lignes de sa table de vérité [7].

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

Exemple $\varphi = ((p \leftrightarrow q) \vee p)$ $[\varphi]_{v4} = 1$ est satisfaite

	p	q	$p \leftrightarrow q$	φ
v_1	**0**	**0**	**0**	**0**
v_2	**0**	**1**	**0**	**0**
v_3	**1**	**0**	**0**	**0**
v_4	**1**	**1**	**1**	**1**

TABLE 1.8 – Exemple de la satisfiabilité

Compatibilité

On généralise la notion de satisfiabilité à un ensemble de formules :
soit $\Sigma = \{ \alpha 1, \alpha 2, ..., \alpha n\}$ un ensemble de formules, Σ est dite compatible si et seulement si étant donné la table de vérité de toutes les formules $\alpha 1, \alpha 2, ..., \alpha n$,il existe au moins une ligne où toutes ces formules sont vraies simultanément.
La compatibilité d'un ensemble de formules est assimilée à la conjonction de toutes ses formules[7].

Remarque

1. Soit E un ensemble de formules et A une formule A$\subseteq$E. alors, si E est compatible, A est satisfiable.

2. L'ensemble vide est satisfiable.
3. L'ensemble de toutes les formules est incompatible.
4. Soit E un ensemble de formules et A une formule A $\subseteq$ E. Alors, si A est insatisfiable alors E est incompatible.

Exemple L'ensemble $\Sigma = \{p \vee q\ ,\ p \wedge q,\ p \rightarrow q\}$ est compatible pour $[\Sigma]_{v4} = 1$.

	p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	Σ
v_1	**0**	**0**	**0**	**0**	**1**	**0**
v_2	**0**	**1**	**0**	**1**	**1**	**0**
v_3	**1**	**0**	**0**	**1**	**0**	**0**
v_4	**1**	**1**	**1**	**1**	**1**	**1**

TABLE 1.9 – Exemple de la compatibilité

Validité (Tautologie)

Une formule α est une tautologie , si et seulement si α est vraie sur toutes les lignes de sa table de vérité.
α est dite invalide si elle est fausse sur au mois une ligne de sa table de vérité[7].
Exemple : La formule $\varphi = ((p \rightarrow q) \vee (q \rightarrow \neg p))$
est une tautologie pour $[\varphi]_{v1} = [\varphi]_{v2} = [\varphi]_{v3} = [\varphi]_{v4} = 1$.

	p	q	$\neg p$	$p \rightarrow q$	$q \rightarrow \neg p$	φ
v_1	**0**	**0**	**1**	**1**	**1**	**1**
v_2	**0**	**1**	**1**	**1**	**1**	**1**
v_3	**1**	**0**	**0**	**0**	**1**	**1**
v_4	**1**	**1**	**0**	**1**	**0**	**1**

TABLE 1.10 – Exemple de la validité (Tautologie)

Antilogie (Contradiction)

Une formule φ est une antilogie (ou : contradiction) ssi pour toute valuation V, $[\varphi]_v = 0$. En termes de table de vérité : la colonne terminale ne contient que des 0.[11]
Exemple $\varphi = ((p \wedge q) \leftrightarrow (q \rightarrow \neg p))$ est antilogie pour $[\varphi]_{v1} = [\varphi]_{v2} = [\varphi]_{v3} = [\varphi]_{v4} = 0$.

	p	q	$\neg p$	$p \wedge q$	$q \rightarrow \neg p$	φ
v_1	**0**	**0**	**1**	**0**	**1**	**0**
v_2	**0**	**1**	**1**	**0**	**1**	**0**
v_3	**1**	**0**	**0**	**0**	**1**	**0**
v_4	**1**	**1**	**0**	**1**	**0**	**0**

TABLE 1.11 – Exemple de l'antilogie

Conséquence

Γ un ensemble de formules tel que : $\Gamma = \{\psi_1, \psi_2, ...\psi_n\}$ et φ est conséquence logique pour $(\Gamma \models \varphi)$ ssi pour toute valuation V, si pour toute formule ψ dans Γ on a $[\psi]_v = 1$, alors $[\varphi]_v = 1$ [11].

Notation $\psi_1, \psi_2, ...\psi_n \models \varphi \equiv \Gamma \models \varphi$

Remarque Nous notons φ est valide par $\models \varphi$, φ est valide si et seulement si φ est conséquence de l'ensemble vide.[17]

Propriété Soient n + 1 formules $\psi_1, ..., \psi_n$ et φ. Soit Γ la conjonction des formules $\psi_1, ..., \psi_n$. Les trois formulations suivantes sont équivalentes :[17]

1. $\psi_1, ..., \psi_n \models \varphi$, c'est-à-dire φ est conséquence des hypothèses $\psi_1, ..., \psi_n$.
2. La formule $\Gamma \rightarrow \varphi$ est valide.
3. $\Gamma \wedge \neg\varphi$ est insatisfaisable(antilogie).

Exemple :$\Gamma = \{A \rightarrow B, A \wedge \neg C, C \vee B\}$, $\Gamma \models A \wedge B$
voir la table 1.12 la conséquence est vérifée par v_7

	A	B	C	$A \to B$	$A \wedge \neg C$	$C \vee B$	$A \wedge B$
v_1	**0**	**0**	**0**	**1**	**0**	**0**	**0**
v_2	**0**	**0**	**1**	**1**	**0**	**1**	**0**
v_3	**0**	**1**	**0**	**1**	**0**	**1**	**0**
v_4	**0**	**1**	**1**	**1**	**0**	**1**	**0**
v_5	**1**	**0**	**0**	**0**	**1**	**0**	**0**
v_6	**1**	**0**	**1**	**0**	**0**	**1**	**0**
v_7	**1**	**1**	**0**	**1**	**1**	**1**	**1**
v_8	**1**	**1**	**1**	**1**	**0**	**1**	**1**

TABLE 1.12 – Exemple de la conséquence

1.3.4 Minimisation de la table de vérité

En général, la méthode de la table de vérité est la méthode la plus naturelle et la plus simple pour obtenir les interprétations d'une formule. Toutefois, elle n'est utilisée que pour les formules simples avec un nombre réduit de variables (problème d'espace mémoire et de temps d'exécution).

Quand le nombre de variables augmente est grand, le nombre total de valuations possibles augmente, donc le nombre de lignes de la table de vérité augmente.
Quand la complexité de la formule soit grand (le nombre des connecteurs logiques),sa nécessite des calcules intermédiaires sur les sous formules donc le nombre des colonnes de la table de vérité accroître (influencer sur le temps d'exécution).

Pour minimiser le temps d'exécution et l'espace de stokage, on utilise l'une des méthodes qui sert à réduire la table de vérité. Parmi d'elle la méthode de diagrame de Quine.

1.4 Diagramme de Quine

En général, les mathématiciens préfèrent d'utiliser la méthode de Quine pour sa simplicité.

1.4.1 Description de la méthode de Quine

Soient A1, A2,.. An des variables propositionnelles de la formule φ :

- On choisit une variable Ai
- On remplace à gauche Ai par 0, et à droite par 1
- On effectue les calculs possibles (pour simplifier sémantiquement φ)
- On répète la procédure pour les formules obtenues jusqu'à ce qu'il n'ait plus de variables propositionnelles [9].

Exemple : $((A \wedge B) \rightarrow (A \wedge C))$

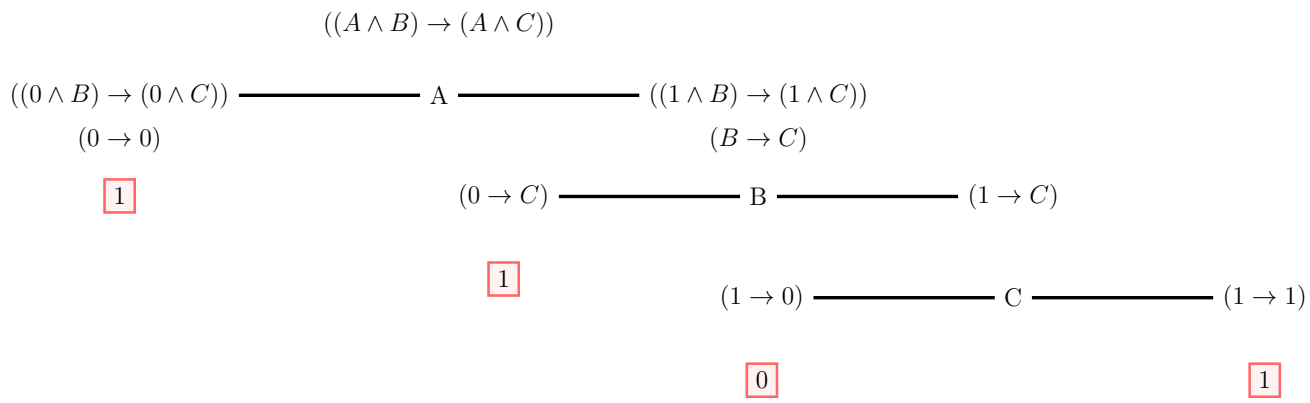


FIGURE 1.4 – Exemple sur la diagramme de Quine

A	B	C	$((A \wedge B) \rightarrow (A \wedge C))$
0	$\forall$	$\forall$	**1**
1	**0**	$\forall$	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

TABLE 1.13 – Table réduite spour $((A \wedge B) \rightarrow (A \wedge C))$

A	B	C	$A \wedge B$	$A \wedge C$	φ
0	**0**	**0**	**0**	**0**	**1**
0	**0**	**1**	**0**	**0**	**1**
0	**1**	**0**	**0**	**0**	**1**
0	**1**	**1**	**0**	**0**	**1**
1	**0**	**0**	**0**	**0**	**1**
1	**0**	**1**	**0**	**1**	**1**
1	**1**	**0**	**1**	**0**	**0**
1	**1**	**1**	**1**	**1**	**1**

TABLE 1.14 – Table complète pour $\varphi = ((A \wedge B) \rightarrow (A \wedge C))$

Cet exemple montre l'utilité du diagramme de Quine, nous avons gagne 4 lignes et 2 colonnes

1.4.2 Algorithme du diagramme de Quine

Algorithm 1: Algorithm Diagramme de quine

```
1  Function diagrammeDeQuine (String formule, Arber arbre) :
       Input: varFix
2      String formuleAfterReplace ;
3      arbre.mettreRacine(formule)
       /* On remplace à gauche,la variable choisie par 0                */
4      formuleAfterReplace ⟵ formule.replaceAll(varFix,0) ;
5      résultat ⟵ caalcule(formuleAfterReplace)
6      if résultat ==0 || résultat == 1 then
7          arbre.mettreSAG(new arbre(résultat)) ;
8      else
9          arbre.mettreSAG(new arbre()) ;
10         diagrammeDeQuine(résultat, arbre.fils_g()
11     end
       /* On remplace à droite,la variable choisie par 1                */
12     formuleAfterReplace ⟵ formule.replaceAll(varFix,1) ;
13     résultat ⟵ calcule(formuleAfterReplace) ;
14     if résultat ==0 || résultat == 1 then
15         arbre.mettreSAD(new arbre(résultat)) ;
16     else
17         arbre.mettreSAD(new arbre()) ;
18         diagrammeDeQuine(résultat, arbre.fils_d()
19     end
20     return arbre ;
21 End Function
```

1.4.3 Implémentation

Dans cette section, nous essayons d'expliquer les interfaces de notre outil, en particulier la méthode de diagramme de Quine après avoir été développée en tant qu'application desktop.

De plus, nous expliquons le matériel et les logiciels de programmation que nous avons utilisés lors de la création de notre application.

1.4.4 Le Matèriel et les logiciels

Le Matèriel

Notre application est implémentée sur pc DELL avec les spécifications suivantes
- Processeur : Processeur Intel(R) Core(TM) i3-7020U à 2,30GHz 2,30 GHz.
- Mémoire installée (RAM) : 4,00Go (3,87Go utilisables).
- Type de système : Système d'exploitation 64bits, processeur x64.
- Système d'exploitation : Windows 10.

Java

Le Java [4] est un langage de programmation orientée objet développé par Sun Microsystems en 1995, et racheté depuis par Oracle. Le principal avantage de Java est son interopérabilité : la technologie fonctionne aussi bien sur Windows que Mac ou Linux, et sur une myriade d'appareils : centres de données, ordinateur, téléphone mobile, lecteur Blu-ray, périphériques TV, consoles de jeux, appareils connectés... Un autre avantage est son caractère universel : le même système peut être utilisé pour une grande variété d'applications. Le langage Java est basé sur le C++, mais avec une approche simplifiée et des fonctionnalités plus avancées .

JavaFX

JavaFX [5] entre en compétition avec les technologies Flex d'Adobe Systems et Silverlight de Microsoft.
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.

JDK

Le kit de développement Java (JDK)[6] est un environnement de développement logiciel utilisé pour développer des applications et des applets Java. Il comprend l'environnement d'exécution Java (JRE), un interpréteur / chargeur (java), un compilateur (javac), un archiveur (jar), un générateur de documentation (javadoc) et d'autres outils nécessaires au développement Java .

Eclipse IDE

Eclipse IDE [3] est un environnement de développement intégré libre (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions .

La spécificité d'Eclipse IDE vient du fait de son architecture totalement développée autour de la notion de plug-in (en conformité avec la norme OSGi) : toutes les fonctionnalités de cet atelier logiciel sont développées en tant que plug-in.

Plusieurs logiciels commerciaux sont basés sur ce logiciel libre, comme par exemple IBM Lotus Notes 8, IBM Symphony ou Websphere Studio Application Developer .

LaTeX

LaTeX[2] est un langage qui permet de créer des documents tout en séparant la forme du fond. Comme pour le HTML, l'interface de rédaction est de type WYSIWYM ("what you see is what you mean" ce que l'on voit est ce que l'on pense) : la forme du document est donc programmée à l'aide de commandes. Le document est généré suite à une compilation qui permet d'aboutir à un fichier au format .ps, .dvi ou .pdf (qui, en l'ocurrence, est le format le plus usité) .

Notre interface principale comprend quatre parties :

L'écran à gauche

Nous avons développé une technique de saisie des connecteurs logiques en utilisant le clavier pour associer chacun des boutons $(+, -, *, /, .)$ à l'un des connecteurs $(\rightarrow, \leftrightarrow, \wedge, \vee, \neg)$ respectivement.

Au centre de l'écran

Cette partie est util quand il n'y a qu'une seule formule à traiter.

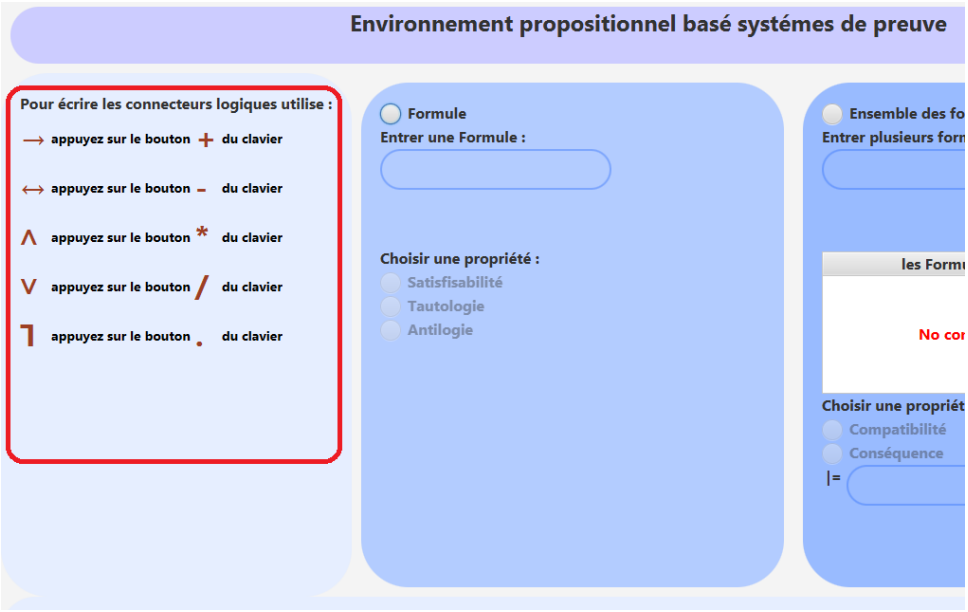


FIGURE 1.5 – Interface principale

1. Afin d'utiliser notre application, selectionner d'abord votre choix(Formule ou Ensemble des formules) :
2. Saisir une formule propositionelle, notre logiciel tient compte aux conditions FBF.
3. Choisir l'une des trois propriètès(Satisfiabiltè, Tautologie, Antilogie).

(Voir figure(1.6))

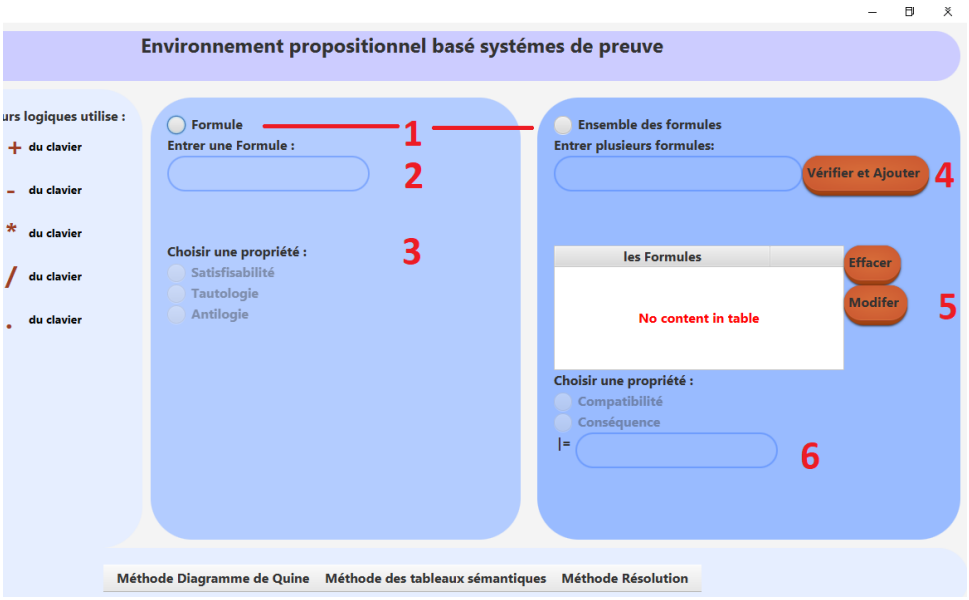


FIGURE 1.6 – Au centre de l'écran

L'écran à droite

Cette partie trait un ensemble des formules.

4 Ecrire une formule puis appuyer sur le bouton (Vérifier et Ajouter) afin de l'être ajouté au tableau.

5 Choisir l'une des deux propriétés (Compatibilité, Conséquence)

6 Si la case de la conséquence est sélectionnée, écrire la formule de la consequence tautologique.

(Voir figure(1.6))

L'écran en bas

Lorsque nous plaçons le pointeur de la souris sur la méthode de Quine, notre logiciel offre trois options : Max, Aléatoire, Choix (Voir figure(1.7)). Nous avons déjà dit dans la

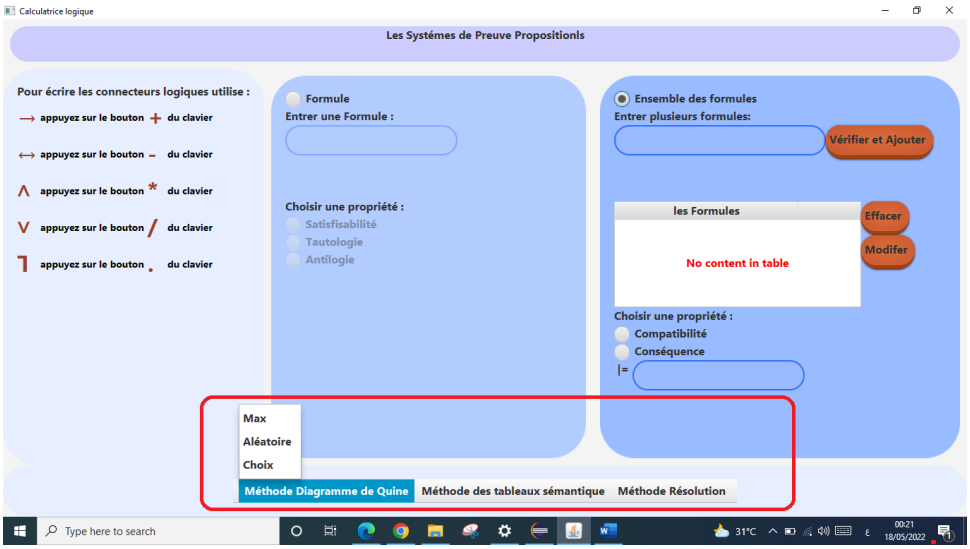


FIGURE 1.7 – Les choix offerps du diagramme de Quine

section précédente que la methode de Quine dépend de choix de la variable propositionelle. Nous avons proposé trois options de choix :

- Le choix aléatoire : la sélection des variables est faite aléatoire ,voir l'exemple de la figure1.8.

- Le choix de max : le programme calcule le nombre d'apparition de chaque variable puis il choisit la variable qui a le nombre le plus grand, le détail dans la figure 1.9.

- Choix de l'utilisateur : l'utilisateur choisie la variable qu'il veut. Voir figure1.10.

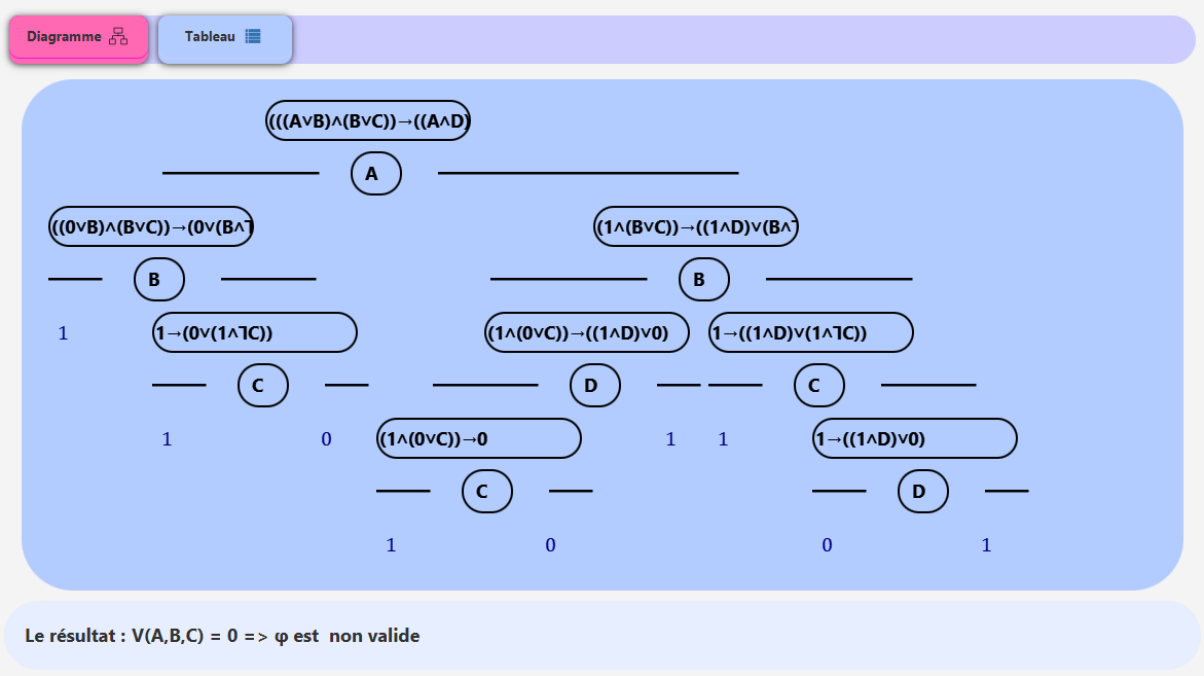


FIGURE 1.8 – Diagramme de Quine par le choix aléatoire

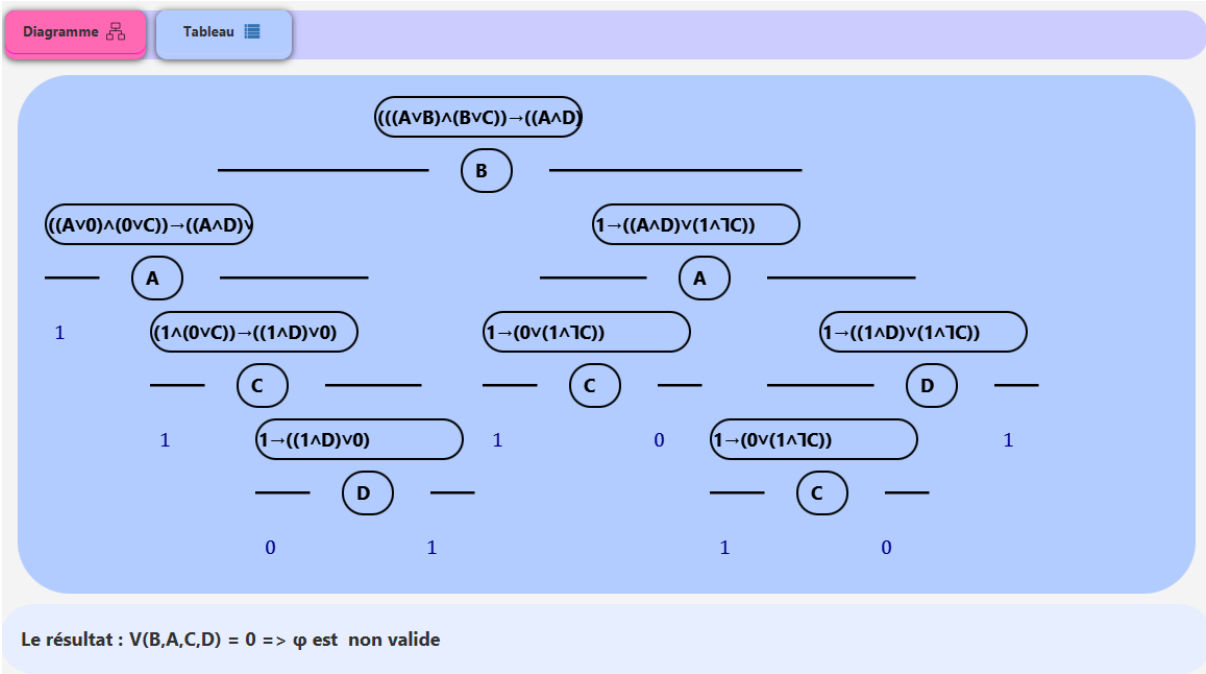


FIGURE 1.9 – Diagramme de Quine par le choix max

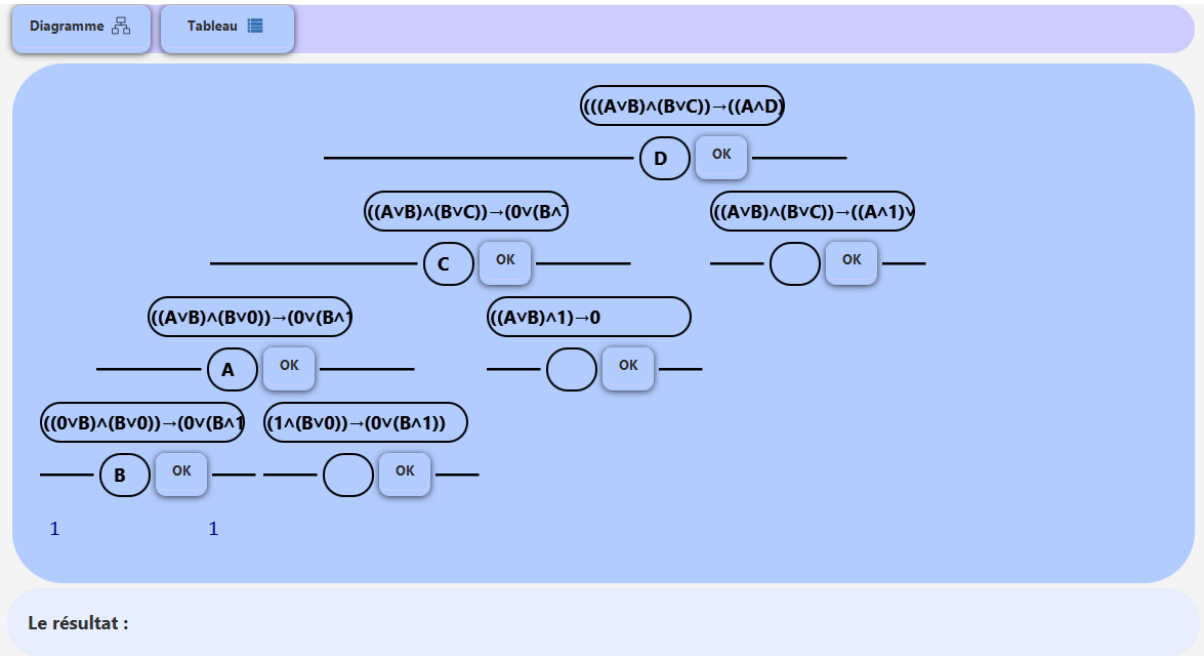


FIGURE 1.10 – Diagramme de Quine par le choix de l'utilisateur

Après voir la construction du diagramme selon l'option choisiée, l'utilisateur peut avoir la table de vérité réduite en cliquant sur le bouton « Tableau ». La figure 1.11 montre deux tableaux, le premier est réduit construit en utilisant le diagramme et le deuxième est un tableau complet inclut tous les modèles.

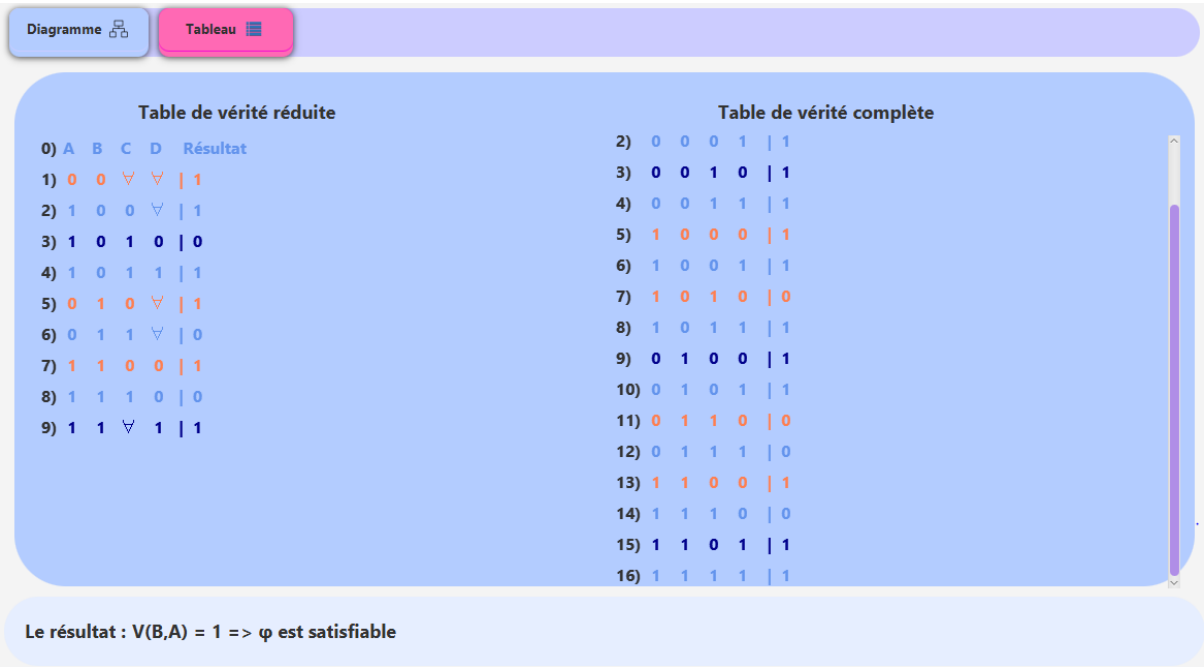


FIGURE 1.11 – Table rèduite versus la table complète

Discusion :

Le principe de la méthode de diagramme de Quine est de fixer une variable, en général le choix de la variable est fait par l'utilisateur, mais son choix n'est pas toujours optimal. Pour celà on a proposé une technique qui permet de choisir la variable la plus present dans la formule donnée, cette technique donne de bon resultas mais elle n'est pas toujours efficace, par exemple (le nombre de repitition de tous les variables est le mêm), aussi l'un de s'est inconvènient, elle donne toujours le mêm résultat. On a amélioré notre application par une autre technique qui fait un choix aleatoire, cette méthode donne beaucoup de résultats pour une seule formule si en l'applique tans de fois, et aussi on peut avoir une solution optimale de ces derniers. Le but voulu de nos techniques proposées est d'optimiser ou bien amèliorer notre application, mais le choix est laissé aux utilisateurs.

1.5 Conclusion

Ce chapitre est composé de deux parties, dans la première partie nous avons présenté le langage propositionnel avec ses deux aspects syntaxique et sémantique, en donnant une explication détaillée avec des exemples. Puis, nous avons proposé un automate finis pour la reconnaissance des formules bien formées.

La deuxième partie présente la méthode du diagramme de Quine en expliquant notre algorithme proposé afin de l'implémenté. Ensuite, nous avons présenté l'aspect applicatif de cette méthode également, nous avons définit le matériel et les logiciels que nous avons utilisés pour l'implémentation.

Dans le prochain chapitre, nous aborderons une autre méthode de preuve, la méthode des tableaux sémantiques.

Chapitre 2

Méthode des tableaux sémantiques

2.1 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 complète, la méthode est coûteuse en espace et en temps, surtout lorsque le nombre de variables est important.

Une preuve courte est préférable à une longue liste de valuations, afin de surmonter ce problème, il est préférable d'utiliser les méthodes de preuve qui aident à réduire le temps, l'espace et surtout elle sert à 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 :

— Axiomatiques (Frege / Hilbert - 1879 à 1934)
— Déduction naturelle (Gentzen - 1935)
— Calcul des séquents (Gentzen - 1935)
— Résolution (Robinson - 1965)
— Tableaux sémantiques (Beth - 1959)
— Méthod Inverse (Maslov - 1968)
— Connexions (Bibel 1974)
— etc...

Nous nous distinguons en mentionnant la méthode des tableaux sémantiques qui nous essayons de la détailler dans ce chapitre.

2.2 Définitions

2.2.1 Présentation de la méthode

C'est une méthode de résolution des problèmes de décision pour le calcul des propositions et les logiques apparentées. La méthode des tableaux peut déterminer la satisfiabilité des ensembles finis de formules de diverses logiques, elle consiste en la recherche systématique d'un modèle d'une formule donnée. Le fait d'imposer une valeur de vérité à une formule peut déterminer univoquement la valeur des composants de cette formule, ou au contraire laisser plusieurs choix possibles. La recherche symantique d'un modèle conduit à la construction progressive d'une arborescence spécifique appelée table sémantique .

2.2.2 Atome

Un atome ou formule atomique est une formule ne comportant qu'une variable propositionnelle (pas de connecteurs)[7]. En d'autre termes, un atome est une proposition

indécomposable. La proposition est une assertion qui a la faculté d'être vrai ou fausse, c'est donc une phrase dout on peut dire sans ambiguitè sa valeur de vèritè.
Example : "un carrè a 4 angles droits" Atome
"Demain, il fera beau" Non Atome

2.2.3 Littérale

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

2.2.4 Complémentaire

- Si A est un atome alors $\{A, \neg A\}$ est une paire de littéraux complémentaires
- Si A est une formule logique alors $\{A, \neg A\}$ est une paire de formules complémentaires Où : A est le complément de $\neg A et \neg A$ est le complément de A.[8]

2.3 Construction de l'arborescence

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

1. Les littéraux
2. Les formules conjonctives
   - La formule $\neg(X \rightarrow Y)$ est conjonctive car elle est équivalente à la conjonction des deux formules (plus simples) X et $\neg$ Y
3. Les formules disjonctives
   - La formule $(X \rightarrow Y)$ est disjonctive car elle est équivalente à la disjonction de $\neg$ X et Y .

La construction des tableaux sémantiques est basée sur les règles de décomposition de formules les règles prolongation (α-règle) et les règles de ramification (β-règle) .

2.3.1 Les règles de prolongation (type α)

On utilise les règles α pour les formules conjonctives, quand on applique une de ces règle à un nœud où on sélectionne une conjonction, alors on obtient un seul fils .

α-Règle d'une formule propositionnelle (**FP**) est équivalente à la conjonction de deux sous-formules $\alpha 1$ et $\alpha 2$ ou à une simplification α. La règle signifie que pour satisfaire une FP il faut satisfaire $\alpha 1$ et $\alpha 2$ [14].

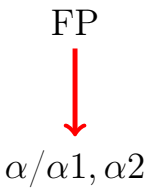


FIGURE 2.1 – Forme de α-règles

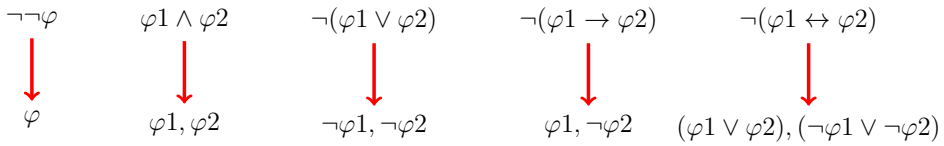


FIGURE 2.2 – α-règles pour la logique propositionnelle

2.3.2 Les règles de ramification (type β)

On utilise les règles β pour les formules disjonctives, quand on applique une de ces règles à un nœud où on sélectionne une disjonction, alors on obtient deux fils.

β-règle d'une FP est équivalente à la disjonction de deux sous-formules $\beta 1$ et $\beta 2$. La règle est notée FP $= \beta 1 \vee \beta 2$ et signifie que pour satisfaire FP, il faut satisfaire $\beta 1$ ou $\beta 2$ [14].

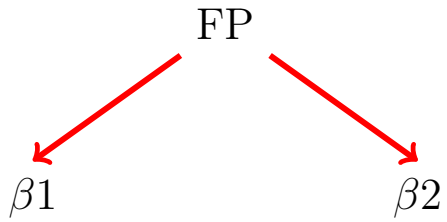


FIGURE 2.3 – Forme de β-règles

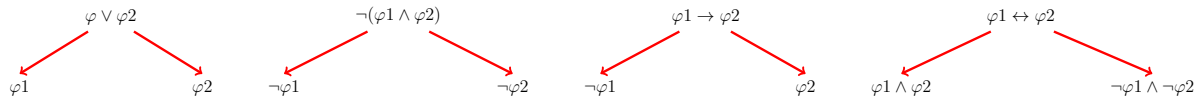


FIGURE 2.4 – β-règle pour la logique propositionnelle

Exemple : dècrit l'utilisation des règles de décomposition avec ses deux type α et β.

$\varphi = ((A \wedge B) \vee \neg C) \rightarrow \neg(D \rightarrow (A \leftrightarrow D)$

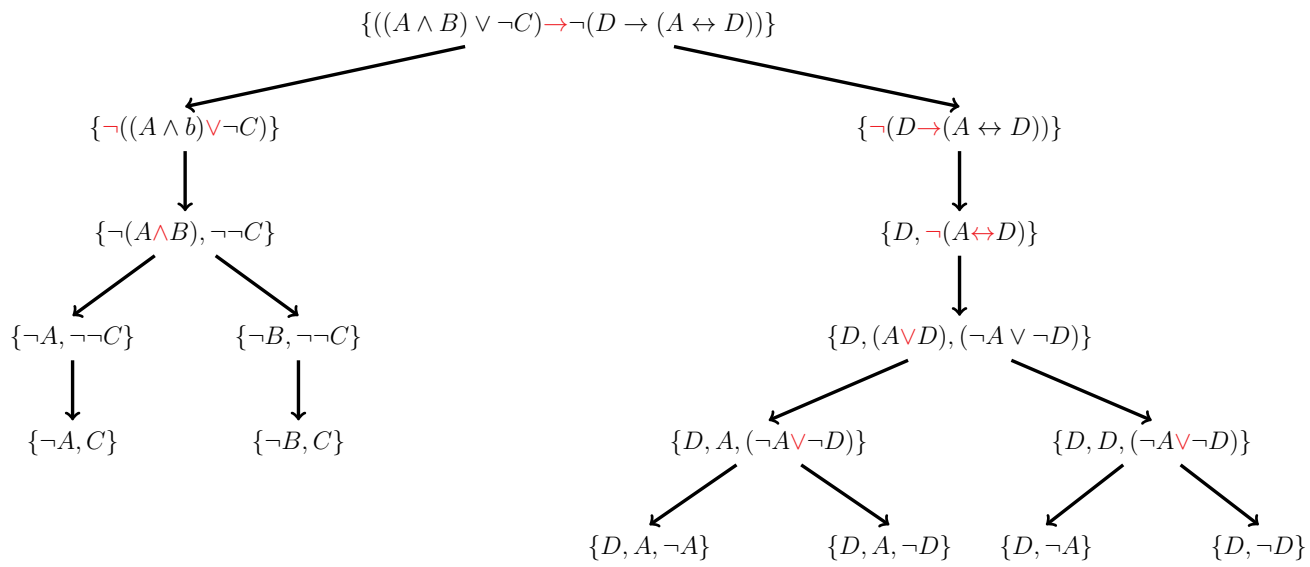


FIGURE 2.5 – Exemple traité par la méthode des tableaux

2.4 Principe général

Le principe général de construction d'un tableau sémantique est illustré par la figure 2.5. Cette dernière est un arbre dans lequel chaque nœud contient un ensemble de formules.
Cet arbre est structuré selon une formule sélectionnée, qu'il s'agisse de règles de prolongation (type α) ou de règles de ramification (type β). Lorsque la construction du tableau est terminée, les feuilles sont des nœuds terminaux qui ne contiennent que des littéraux. La lecture de l'arbre est trés simple, on dit qu'un nœud est satisfaisable si et seulement s'il ne contient pas deux littéraux complémentaires dans la même feuille, et on dit qu'il insatisfaisable si et seulement il contient la littérale et la complémentaire dans la même feuille, c'est la cas de l'example traitè.

Tableau fermé : un tableau est dit fermé si toutes ses feuilles sont insatisfaisables.
Tableau ouvert : Un tableau est dit ouvert si au moins une feuille est satisfaisante
Tableau complet : un tableau sémantique est dit complet si toutes ses feuilles sont des feuilles fermées ou des feuilles ouvertes.

Exemple : ce tableau 2.5 est complet ouvert car il contient au moins une feuille ouverte $\{\neg B, \neg C\}$.

2.4.1 Le tableau sémantique en pratique

Preuve de la satisfiabilité

Pour prouver qu'une formule propositionnelle est satisfaite on cherche systématiquement un modèle. Le problème de preuve de satisfiabilité a été réduit à un problème de

satisfiabilité d'un ensemble de littéraux [13].
Donc : Si T(φ) est ouvert, alors φ est satisfiable.
Si T(φ) est fermé, alors φ est insatisfiable (antilogie).
Exemple : Preuve de la satisfiabilité de φ.
$\varphi = ((A \vee B) \wedge \neg A)$.

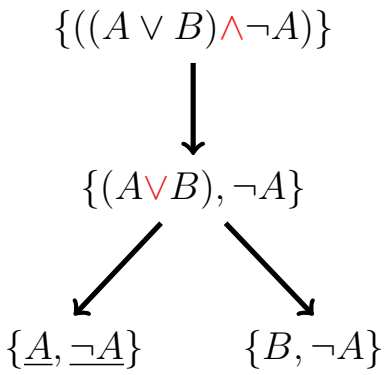
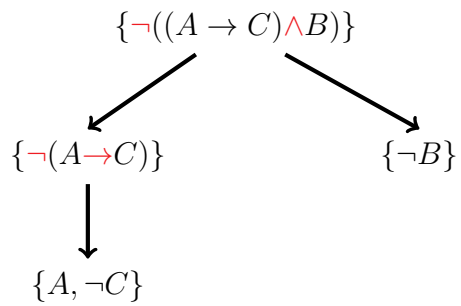


FIGURE 2.6 – Exemple de preuve de la satisfiabilité

v(B,A) = (1,0) ⇒ T(φ) est ouvert ⇒ φ est satisfiable

Preuve de la validité

Pour prouver qu'une formule est valide (ou tautologie) revient à prouver l'inconsistance de sa négation. φ valide → $\neg\varphi$ est non satisfaite [15].
Donc : Si T($\neg\varphi$) est fermé, alors φ est valide.
Si T($\neg\varphi$) est ouvert alors,φ est non valide.
Exemple : Prouver que $\varphi 1$ et $\varphi 2$ est valide $\varphi 1 = ((A \to C) \wedge B)$
$\varphi 2 = ((A \to C) \vee (A \wedge \neg C))$



v(A,C)= (1,0) ⇒ T($\varphi 1$) est ouvert ⇒ $\varphi 1$ est non valide.

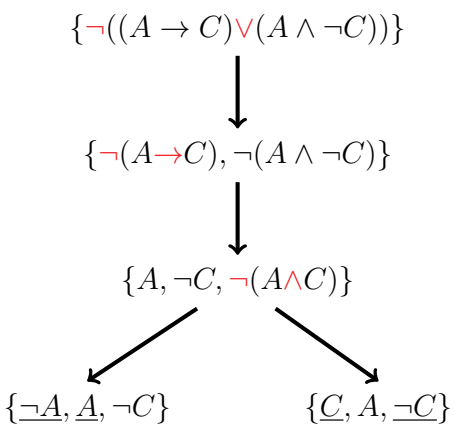


FIGURE 2.7 – Exemple de preuve de la validité(ou tautologie)

T($\neg\varphi 2$) est fermé $\Rightarrow$ $\varphi 2$ est Valide.

Preuve de la compatibilité d'un ensemble de formules

Prouver que $\Sigma = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ est compatible revient à prouver que la formule $\varphi = \varphi_1, \varphi_2, \ldots, \varphi_n$ est satisfaite [15].
Donc : Si T(φ) est ouvert alors φ est compatible.
Si T(φ) est fermé alors φ est incompatible
Exemple : Prouver que l'ensemble $\Sigma = \{(B \to C), \neg(A \to B)\}$ est compatible.

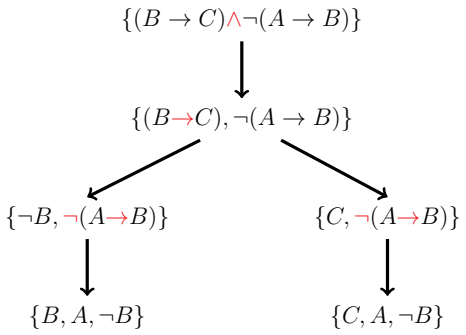


FIGURE 2.8 – Exemple de preuve de la compatibilité

T($(B \to C) \wedge \neg(A \to B)$) est ouvert alors Σ est compatible.

Preuve de la conséquence

Prouver que $\varphi_1, \varphi_2, \ldots, \varphi_n \models \varphi$ est une conséquence logique revient à prouver que la formule FP $\equiv \varphi_1, \varphi_2, \ldots, \varphi_n \to \varphi$ est valide (tautologie) [15].
Exemple : Montrer que $\{(B \to C), \neg(A \to B)\} \models (A \wedge \neg B)$.

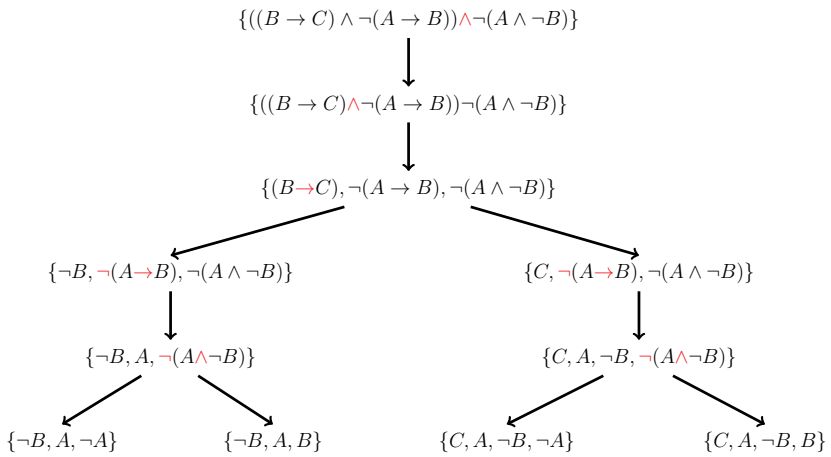


FIGURE 2.9 – Exemple de preuve de la conséquence

T$((B \to C) \wedge \neg(A \to B)) \wedge \neg(A \wedge \neg B))$ est fermé alors $(A \wedge \neg B)$ est conséquence $\{(B \to C), \neg(A \to B)\}$.

2.4.2 Propriétés de la méthode des tableaux sémantiques

L'utilisation de tableaux sémantiques a pour but de montrer la satisfiabilité d'une formule ou la compatibilité d'un ensemble de formules.

— si T(A) est fermé, alors A est non satisfaite (antilogie)
— si T(¬B) est fermé, alors B est valide
— si A est satisfiable, alors T(A) est ouvert
— si B n'est pas valide, alors T(¬B) est ouvert

Prouver la complétude revient à prouver la réciproque, c'est-à-dire l'un des énoncés suivants :

— si A est antilogie, alors T(A) est fermé .
— si B est valide, alors T(¬B) est fermé .
— si T(A) est ouvert, alors A est satisfiabl .
— si T(¬B) est ouvert, alors B n'est pas valide.
— si Γ est un ensemble fini de formules tel que $\Gamma = \{\psi_1, \psi_2, \psi_3, ..., \psi_n\}$.
T($\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n$) est ouvert, alors Γ est compatible .
— si $\Gamma = \{\psi_1, \psi_2, \psi_3, ..., \psi_n\}$, $\Gamma \models \varphi \Rightarrow \{\psi_1, \psi_2, \psi_3, ..., \psi_n\} \models \varphi$
T($\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n \wedge \neg\varphi$) est fermé, alors φ est conséquence tautologique de Γ.

2.5 Implémentation de la méthode des tableaux

Dans cette section nous allons mettre en évidence l'aspect applicatif de la méthode des tableaux sémantiques, en présentant notre algorithme expliquant les différentes in-

terfaces de notre système à l'aide des exemples pour illustrer les propriétés logiques.

2.5.1 Algorithme de la méthode des tableaux sémantiques

Notre idèe principale est de créer un arbre dont les noeuds sont des listes afin de grouper les formules propositionelles. Premièrment, notre algorithme choisit le connecteur principal, la prioritè des conecteurs est connue par les délimiteurs propositionnels. Deuxiement l'algorithme selectionne la règle adèquate(prolongation ou ramification) selon le connecteur choisi, si une règles de prolongation est traitè l'algorithme criée un seul fils, sinon deux fils. L'algorithme est rèpètè pour les noeuds obtenus jusqu'à ce qui'l n'ait plus de formules composèes(chaque feuille a seulment des littèraux).

Algorithm 2: Algorithme Tableaux Sémantiques

```
22 Function tableauxSémantique(Arbre arbre , int len, int pos) :
23   if arbre ≠ null and Not(arbre.arbreVide()) and pos <= len then
       /* L'ensemble des formules                                    */
24     Liste list ⟵ (Liste)arbre.racine() ;
25     ListelistGouch;
26     ListelistDroit;
27     Arbre arbreDeComposition ⟵ connecteurPrincipal(list.ieme(pos)) ;
28     connecteurPrincipal ⟵ arbreDeComposition.racine() ;
29     if connecteurPrincipal = ¬ then
30       éliminationNégation(list.ieme(pos)) ;
31       tableauxSémantique( arbre, len , pos) ;
32     else
33       listGouch ⟵ list.copier() ;
34       listGouch.supprimer(pos) ;
         /* appliqué α-règle                                         */
35       if connecteurPrincipal = ∧ then
36         listGouch.inserer(arbreDeComposition.fils_g().racine(), pos)
37         listGouch.inserer(arbreDeComposition.fils_d().racine(), pos+1) ;
38         arbre.mettreSAG(new Arbre(listGouch)) ;
39         tableauxSémantiques(arbre.fils_g(), pos, len+1) ;
40       else
           /* Appliqué β-règle                                       */
41         listDroit ⟵ listGouch.cpier() ;
42         if connecteurPrincipal = ∨ then
43           listGouch.inserer( arbreDeComposition.fils_g().racine(), pos) ;
44           listDroit.inserer(arbreDeComposition.fils_d().racine(), pos) ;
45         else if connecteurPrincipal =→ then
46           listGouch.inserer("¬("+arbreDeComposition.fils_g().racine()+")",
               pos) ;
47           listDroit.inserer(arbreDeComposition.fils_d().racine(), pos) ;
48         else if connecteurPrincipal =↔ then
49           listGouch.inserer(arbreDeComposition.fils_g().racine()∧arbreDeComposition.fil
50           listDroit.inserer(¬arbreDeComposition.fils_g().racine() ∧
               ¬arbreDeComposition.fils_d().racine(),pos) ;
51       end
52       arbre.mettreSAG(new Arbre(listGouch)) ;
53       arbre.mettreSAD(new Arbre(listDroit)) ;
54       tableauxSémantiques(arbrefils_g(), pos, len) ;
55       tableauxSémantiques(arbrefils_d(), pos, len) ;
56     end
57   end
58   return arbre ;
```

2.5.2 Interfaces de l'outil

Notre objectif est de répondre au désir de l'utilisateur par l'utilisation des tableaux sémantiques en fonction des options qu'il saisit.

Comme nous savons déjà que lorsqu'il n'y a qu'une formule, trois propriétés logiques peuvent traitées, la satisfiabilité et la validité et antilogie. Et lorsqu'il y a un ensemble de formules, il n'y a que deux propriétés logiques la compatibilité et la conséquence.

Nous allons essayer de les prouver par notre système.

Preuve de la satisfiabilité

Si l'utilisateur saisit une seule formule et choisit la satisfiabilité . On cherche au moins une solution.

Donc :

Si T(φ) est ouvert alors T(φ) est satisfiable.

Si T(φ) est fermé alors φ est insatisfiable.

Exemple $\varphi = ((((a \wedge b) \vee \neg b) \leftrightarrow d) \rightarrow \neg c)$ À partir de la figure 2.11, nous remarquons

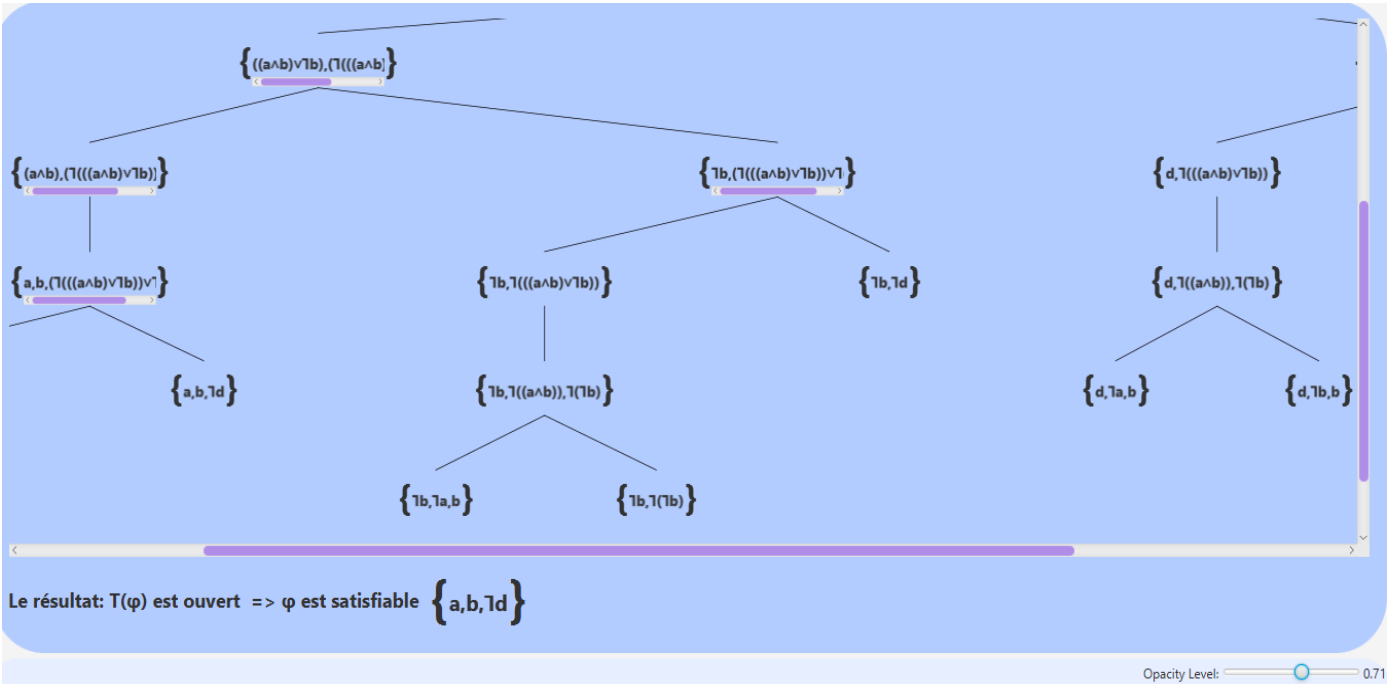


FIGURE 2.11 – Exemple de tableau sémantique pour la satisfiabilité

qu'il existe des feuilles ouvertes, par exemple { a , b, $\neg d$ } donc le tableau T(φ) est ouvert, la formule φ est satisfaisante par le model V(a , b, $\neg d$)=(1,1,0).

Remarque noter que si le tableaux sémantique est grand, on peut réduire et agrandir l'écran pour que toutes les feuilles soient lisibles.

Preuve de la validité (tautologie)

Dans le cas où l'utilisateur choisit la propriété de la validité, il faut verifie l'ouverture de toutes les feuilles, et en plus calculer le nombre des modeles qui doit etre égal au 2^n tel que n est le nombre global des variables propositionelles de la formule traitè. Afin de minimiser la complexitè algorithmique, il vaut mieux prouver que son complément est non satisfaite(antilogie).

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

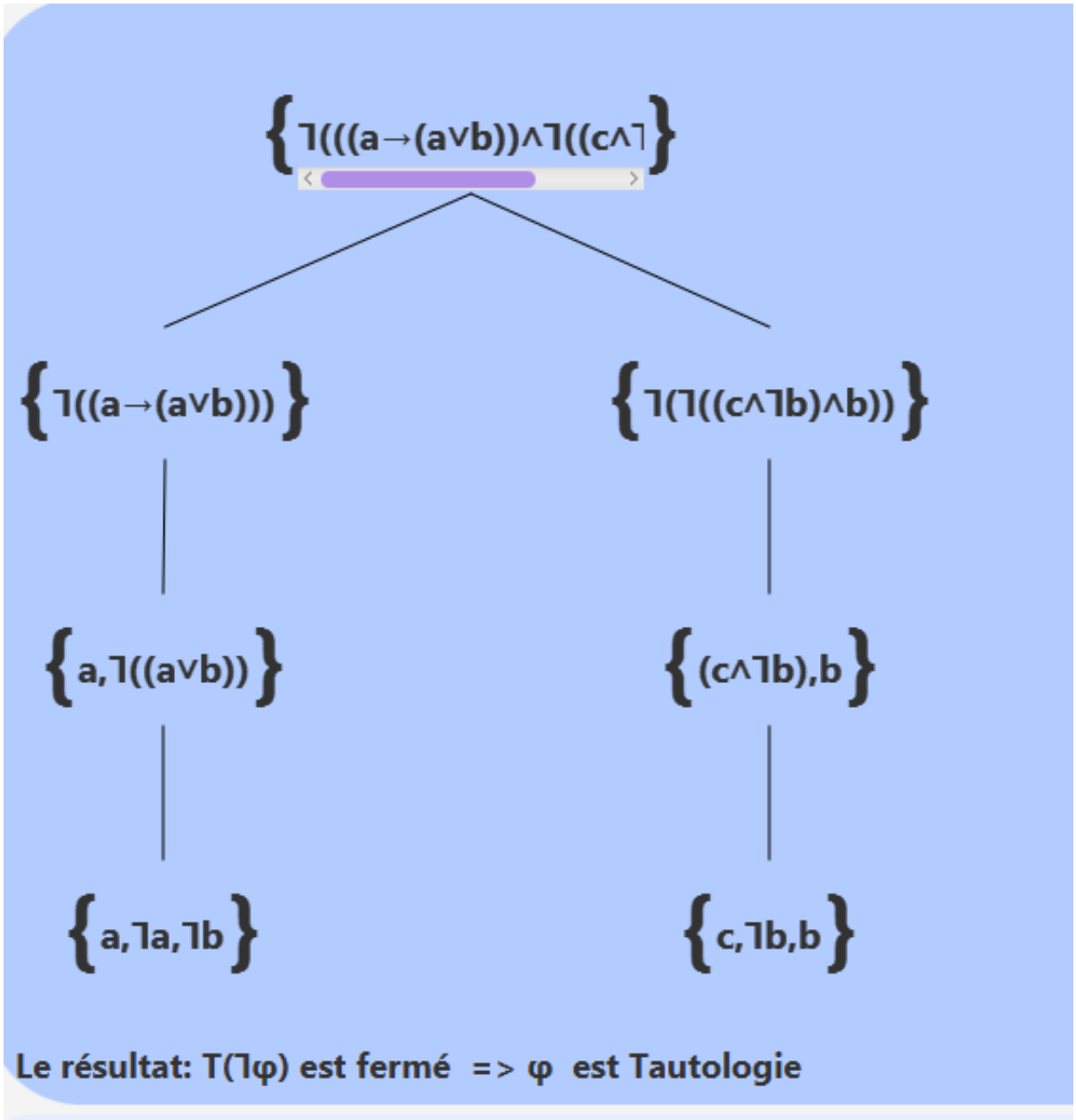


FIGURE 2.12 – Exemple de tableau sémantique pour la validité (tautologie)

On remarque que T($\neg\varphi$) est fermé alors φ est valide (tautologie).

L'antilogie

Dans le cas où l'utilisateur choisit la propriété de l'antilogie, il faut que toutes les feuilles de la table adèquate de φ saisie doivent être fermées.
Si φ est antilogie, alors T(φ) est fermé
Exemple $\varphi = ((\neg(A \to \neg B) \wedge ((C \vee D) \vee B)) \wedge ((A \wedge C) \wedge \neg(D \vee B)))$

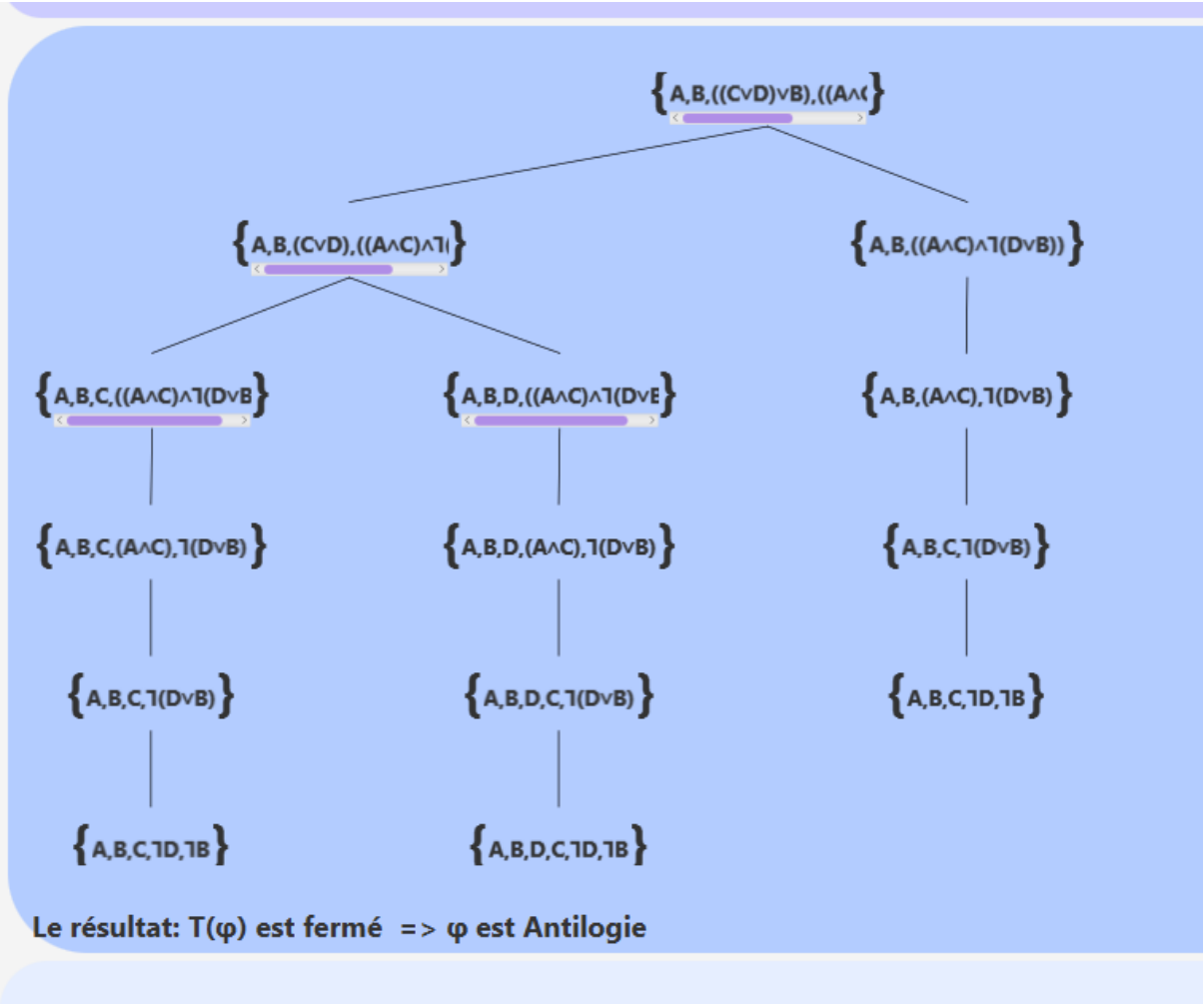


FIGURE 2.13 – Exemple de tableau sémantique pour l'antilogie

La compatibilité

Il s'agit d'un ensemble limité de formules. Chaque formule est insérée dans le tableau après avoir vérifiée(Sous les contraintes FBF), puis l'utilisateur choisit case de la compatibilité.
Afin de prouver la compatibilitè d'un ensemble de formules il faut étudier la satisfabilitè de leur conjonction. Si cette nouvelle formule est satisfiable, alors nous concluons que cet ensemble est compatible, nous avons :
$\Gamma = \{\psi_1, \psi_2, \psi_3, ..., \psi_n\}$

$\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n = \varphi$
Si T(φ) est ouvert alors Γ est compatible sinon Γ est incompatible.
Exemple $\Gamma = \{((a \wedge b) \vee (c \wedge b)), (c \wedge d)), ((c \vee d) \wedge d), ((a \rightarrow b) \leftrightarrow (a \wedge b))\}$

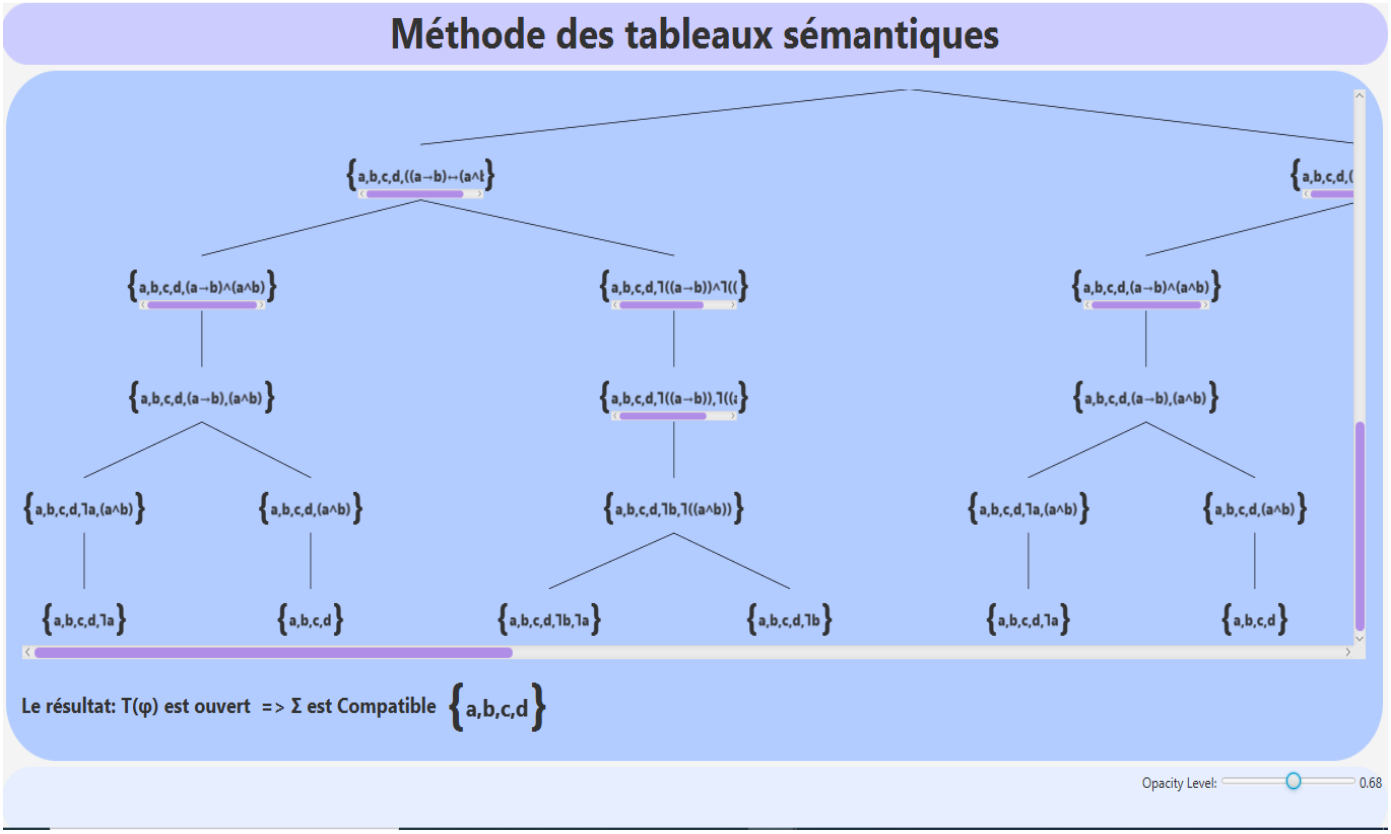


FIGURE 2.14 – Exemple de tableau sémantique pour la compatibilité

On remarque que {a,b,c,d} est une feuille ouverte donc T(Γ) ouvert $\rightarrow$ Γ est un ensemble compatible.

La conséquence

On peut étudier la consequence entre deux formules ou entre un ensemble de formules, notre outil offre deux possibilités . Nous effsyons d'expliquer la conséquence d'un ensemble, dans ce cas l'utilisateur insert l'ensemble des formule dans le tableau de la figure 2.15, apres il doit taper la formule conséquence en dessous dans le champ de saisie qui lui est assigné.
Pour prouver la conséquence, le systeme etudier la non satisfabilité de la conjonction de l'ensemble et du complément de la formule consèquence. $\Gamma = \{\psi_1, \psi_2, \psi_3, ..., \psi_n\} \models \varphi$
$\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n \wedge \neg\varphi = \psi$
si T(ψ) est fermé alors Γ est conséquence sinon pas conséquence .
Exemple $\Gamma = \{\neg(a \to b), (a \wedge (b \vee c)), (a \leftrightarrow b)\}$
prouver que $\Gamma \models a$.

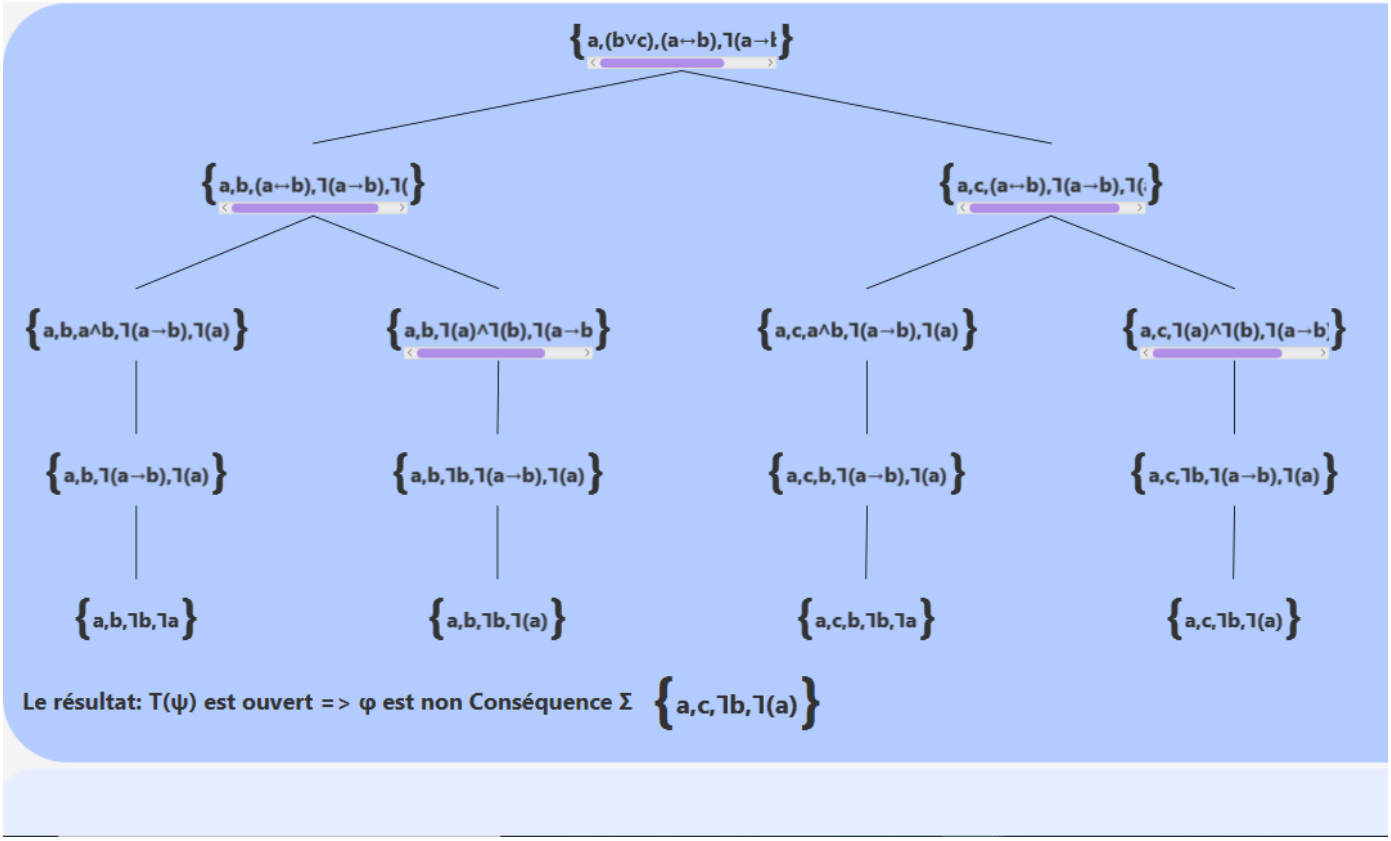


FIGURE 2.15 – Exemple de tableau sémantique pour la conséquence

T($((a \to b) \wedge (a \wedge (b \vee c)) \wedge (a \leftrightarrow b) \wedge \neg a)$ est fermé
(Toutes le feuilles ont des lettéraux complémentaires), alors la ounséquence existe $\Gamma \models a$.

2.6 Conclusion

Ce chapitre est divisé en deux sections, une section théorique et une section pratique. La première section introduit une présentation générale de la méthode des tableaux sémantiques avec certaines définitions essentiels, atome, littérale, complémentaire, etc. et leurs règles principales de la décomposition. Egalement, la section explique comment prouver les propriétés de la logique en utilisant cette méthode.

En ce qui concerne la deuxième section, nous avons présenté notre algorithme proposé pour le développement de l'application en donnant diverses interfaces pour une meilleure explication.

Dans le prochain chapitre, nous discuterons la méthode de preuve, résolution par réfutation.

Chapitre 3

Résolution par réfutation

3.1 Introduction

Il existe de nombreuses méthodes de preuve, chacune se distingue par son propre principe, et parmi d'elles nous avons déjà présenté la méthode des tableaux sémantiques Dans ce chapitre nous allons étudier l'une des techniques qui est la plus souvent utilisée dans les programmes de démonstration automatique, la résolution. 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 un certain type spécifique des formules, elles sont appelées formes clausales.

3.2 Formes Normales (FN)

Réellement, il existe plusieurs possibilités pour exprimer une formule dans une forme équivalente. En fait, pour toute formule φ il y a une infinité de formules équivalentes : φ,$\varphi \wedge \varphi \wedge \varphi$,$\varphi \wedge \varphi \wedge \varphi$,...,etc. D'autres pour lesquelles l'équivalence peut être beaucoup moins évidente. Vraiment, nous n'avons pas besoin de toute cette multitude de possibilités d'exprimer la même chose. Généralement, quand les informaticiens travaillent avec des expressions symboliques, ils définissent souvent un tel standard, appelé une «forme normale». Dans cette section, nous allons étudier quelques formes normales des formules propositionnelles.

3.2.1 Clause :

Une clause est un littéral ou une disjonction de littéraux (disjonction élémentaire).[7]

Exemple

- $a \vee b \vee \neg c$
- A
- $\neg$ B
- ϕ ou $\perp$ est la clause vide

3.2.2 Monôme :

Un monôme conjonctif est un littéral ou une conjonction de littéraux (conjonction élémentaire). [7]

Exemple

- $a \wedge b \wedge \neg c$ est un monôme
- A
- $\neg$ B
- ϕ

3.2.3 Forme Normale Négative (FNN) :

Une formule est en forme normale de négation si elle est construite avec les connecteurs $\neg, \vee et \wedge$, seulement , et elle ne contient pas d'applications de l'opérateur $\neg$ sauf des applications devant les variables propositionnelles.

Exemple

- $((a \wedge b) \vee \neg c))$
- A123
- $\neg$ B
- $\neg(a \wedge b)$ n'est pas FNN
- (a $\rightarrow$b) n'est pas FNN

3.2.4 Forme Normale Conjonctive (FNC) :

Une formule est en forme normale conjonctive si elle est un clause ou une conjonction de clauses[7].

Exemple

- $(a \vee b) \wedge (a \vee b \vee \neg c) \wedge \neg c$
- $a \vee b \vee \neg c$
- $A123 \wedge (\neg b \wedge c)$
- A123

Ils sont des FNC

- $(a \vee b) \wedge (a \vee b \wedge \neg c) \wedge \neg c$
- $(a \vee b) \wedge \neg(b \vee \neg c)$
- $A123 \vee (\neg b \wedge c)$
- $\neg\neg A123$

Ils ne sont pas des FNC

Mise sous forme normale conjonctive (FNC) :

D'une manière analogue, on obtient la FNC à partir de la table de vérité. Soit φ une formule du calcul propositionnel contenant N variables $A_1, A_2, ..., A_n$. Supposons que Q est la table de vérité de φ .
La FNC de φ est obtenue de la manière suivante : [7]

- A1,A2,...,A_n sont des littéraux.
- Soit P le nombre de lignes de Q telles que V (φ) = 0.

- Pour chaque ligne P_i , où $\vee(\varphi) = 0$.

$$\text{on détermine } Q_i = \text{A1} \vee \text{A2} \vee ... \vee A_n \text{ telles que } \begin{cases} A_i & \text{Si v}(A_i)=0 \\ \neg A_i & \text{Si v}(A_i)=1 \end{cases}$$

- Q $= Q1 \wedge Q2 \wedge ... \wedge Q_n$.

Exemple : $\varphi = ((A \rightarrow B) \wedge C)$
Ecrire φ sous la forme de FNC. FNC de φ est $(A \vee B \vee C) \wedge (A \vee \neg B \vee C) \wedge (\neg A \vee B \vee C) \wedge (\neg A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee C)$

A	B	C	$A \rightarrow B$	φ
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

TABLE 3.1 – Exemple de FNC

3.2.5 Forme Normale Disjonctive (FND) :

Une formule est en forme normale disjonctive si elle est un monôme ou une disjonction de monômes [7].

Exemple

— $(a \wedge b) \vee (a \wedge b \wedge \neg c) \vee \neg c$
— $a \wedge b \wedge \neg c$
— $d123 \vee \neg b \vee c$
— D123

Ils sont des FND

— $(a \wedge b) \vee (a \vee b \wedge \neg d) \vee \neg c$
— $(a \vee b) \wedge \neg(b \vee \neg d)$
— $(A123 \vee \neg b) \wedge d$

— $\neg\neg D123$

Ils ne sont pas des FND

Mise sous forme normale disjonctive (FND) :

Il est nécessaire d'avoir un moyen algorithmique pour obtenir la (FND) à partir de la table de vérité. Soit φ une formule du calcul propositionnel contenant N variables $A_1, A_2, ..., A_n$.. Supposons que Q est la table de vérité de φ .

La FND de φ est obtenue de la manière suivante : [7]

- A1,A2,...,A_n sont des littéraux
- Soit P le nombre de lignes de Q telles que V $(\varphi) = 1$.
- Pour chaque ligne P_i, où V $(\varphi) = 1$

$$\text{on détermine } Q_i = \text{A1} \wedge \text{A2} \wedge ... \wedge A_n \text{ telles que } \begin{cases} A_i & \text{Si v}(A_i)=1 \\ \neg A_i & \text{Si v}(A_i)=0 \end{cases}$$

- Q $= Q1 \vee Q2 \vee ... \vee Q_n$.

Exemple : $\varphi = ((A \rightarrow B) \wedge C)$

Ecrire φ sous la forme de FND.

A	B	C	$A \rightarrow B$	φ
0	**0**	**0**	**1**	**0**
0	**0**	**1**	**1**	**1**
0	**1**	**0**	**1**	**0**
0	**1**	**1**	**1**	**1**
1	**0**	**0**	**0**	**0**
1	**0**	**1**	**0**	**0**
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**

TABLE 3.2 – Exemple de FND

FND de φ est $(\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge B \wedge C)$

3.3 Résolution

En logique mathématique, la règle de résolution ou principe de résolution de Robinson est une règle d'inférence logique qui généralise le modus ponens. Cette règle est principalement utilisée dans les systèmes de preuve automatiques, elle est à la base du langage de programmation logique.

3.3.1 Ensemble de clauses

Un ensemble de clauses, par exemple : $\{p \vee q \vee \neg r, \neg p \vee \neg q \vee \neg r, p \vee q, p, \neg p\}$ est considéré comme la conjonction de toutes les clauses qu'il contient[1].
Attention : Ne pas confondre entre la clause vide, qui est $\perp$, et l'ensemble vide de clauses, qui est $\top$ (le vrai) [1].

3.3.2 Résolvante de deux clauses

Soit deux clauses, C1 et C2, telles que le littéral A figure dans l'une et le littéral $\neg A$ figure dans l'autre (par exemple : A $\in$ C1, $\neg A \in$ C2), on appelle résolvante de C1 et C2 la clause obtenue en supprimant le littéral A de l'une et le littéral $\neg A$ de l'autre et en réunissant tout ce qui reste en une seule clause [1].

- Elle est appelée clause résolvante ou résolvant de C1 et de C2.
- A et $\neg A$ sont les littéraux résolus.

Cas possibles en résolution :

- Si C1 =$\neg p \vee q \vee r$ et C2= $p \vee q \vee s$, alors Résolvant (C1, C2) = $q \vee r \vee s$ (on supprime aussi en même temps les littéraux redondants).
- Si C1 =$\neg p \vee q \vee r$ et C2= $p \vee \neg q \vee s$ alors Résolvant (C1, C2) = $q \vee \neg q \vee r \vee s$ ou Résolvant (C1, C2) = $p \vee \neg p \vee r \vee s$
- Si C1 =p et C2 =$\neg p$ Résolvant (C1, C2) = $\perp$

3.3.3 Démonstration :

Soient A, B des formules et X un littéral et soit v une interprétation.
Supposons v$(A \vee X) = 1$ et v$(B \vee \neg X) = 1$.

$$\begin{cases} \text{Si v(X)} = 1 \quad \text{alors v(B)} = 1 \text{ et donc v(A} \vee \text{B)} = 1. \\ \text{Si v(X)} = 0 \quad \text{alors v(A)} = 1 \text{ et donc v(A} \vee \text{B)} = 1. \end{cases}$$

En conclusion : $(A \vee X), (B \vee \neg X) \models (A \vee B)$.
Cette régle trés simple est appelée régle de résolution dans le cas où X est une proposition et où A, B sont des clauses[14].

3.3.4 Réfutation

Une réfutation d'un ensemble de clauses C est une preuve de la clause vide (Ø) à partir de C[16].
Notation $C \models \perp$.

3.3.5 Résolution par réfutation

A partir d'un ensemble de clauses contenant la négation de la proposition à démontrer applique 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 $\perp$ si possible [1].
Exemple : Soit Σ l'ensemble de clauses $\{\neg p \vee q, p \vee \neg q, \neg p \vee \neg q, p \vee q\}$.
Nous montrons que $\Sigma \models \perp$:

1	$p \vee q$	Hypothèse
2	$p \vee \neg q$	Hypothèse
3	p	Résolvant de 1,2
4	$\neg p \vee q$	Hypothèse
5	q	Résolvant de 3,4
6	$\neg p \vee \neg q$	Hypothèse
7	$\neg p$	Résolvant de 5,6
8	$\perp$	Résolvant de 3,7

Stop ! On a trouvé $\perp$ donc $\Sigma = \{\neg p \vee q, p \vee \neg q, \neg p \vee \neg q, p \vee q\}$ est incompatible et la formule qui joint les clauses de cet exemble est antilogie (insatisfiable/inconsistante).
On peut formuler ce qui précède sous forme d'un algorithme.

3.3.6 Algorithme de résolution

Pour prouver que la validité φ,il faut prouver que le complément de φ ($\neg\varphi$) est antilogie, et cela se fait en écrivant $\neg\varphi$ sous la forme d'un ensemble des clauses et on applique la méthode de résolution par réfutation à cet ensemble.

Exemple : $\varphi = ((A \rightarrow B) \wedge \neg C) \vee (B \rightarrow C)$, prouver que φ est vrai en utilisant la méthode de résolution par réfutation.

1. Négation φ : $\neg\varphi = \neg(((A \rightarrow B) \wedge \neg C) \vee (B \rightarrow C))$
2. Extraction des clauses : $\Sigma = \{(A \vee C), (\neg B \vee C), B, \neg C\}$
3. Appliquer la méthode résolution par réfutation à Σ :

Stop ! On a trouvé $\perp \Rightarrow \neg\varphi$ est antilogie donc φ est tautologie.

Algorithm 3: Algorithme de résolution

Input : Ecrire la négation d'une formule φ ;
Mettre $\neg\varphi$ sous forme d'un ensemble de clauses ;
while *e la clause vide n'est pas rencontrée et qu'il existe des paires réductibles* **do**
    Chercher des clauses résolvantes ;
    Ajouter ce résultat à la liste des clauses ;
end
if *on trouve la clause vide* **then**
    φ est valide ;
else
    φ est non valide ;
end

FIGURE 3.1 – Algorithme de résolution

1	$A \vee C$	Hypothèse
2	$\neg B \vee C$	Hypothèse
3	B	Hypothèse
4	$\neg C$	Hypothèse
5	C	Résolvant de 2,3
6	$\bot$	Résolvant de 4,5

3.4 Implémentation de la méthode résolution

Dans cette section nous allons mettre en évidence l'aspect applicatif de la méthode de résolution, et comment préparer des formules pour que cette méthode fonctionne. Ensuite, nous allons expliquer comment l'appliquer aux différentes propriétés logiques.

3.4.1 Étapes de traitement d'une formule

La formule passe par plusieurs étapes jusqu'à ce qu'elle se présente sous la forme d'un ensemble de clauses.

Arbre de décomposition

chaque formule traitée doit être décomposé en trois parties, le connecteur logique principale, sous-formule droite et sous-formule gauche, cela si le connecteur binaire. Sinon la négation et la sous-formule gauche.

Exemple :

$\varphi = (\neg(A \wedge B) \rightarrow (C \leftrightarrow D))$

le connecteur logique principal est $\rightarrow$
la sous-formule de droite est $C \leftrightarrow D$
la sous-formule de gauche est $\neg(A \wedge B)$

1. Nous mettons le connecteur principal à la tête de l'arbre
2. Nous plaçons la sous-formule droite à droite de l'arbre et la sous-formule gauche à gauche de l'arbre
3. Nous répétons le processus de décomposition des sous-formules $(C \leftrightarrow D)$ et $(\neg(A \wedge B))$ jusqu'à atteindre les variables.

Exemple :$\varphi = (\neg(A \wedge B) \rightarrow (C \leftrightarrow D))$
Créer Arbre de décomposition φ.

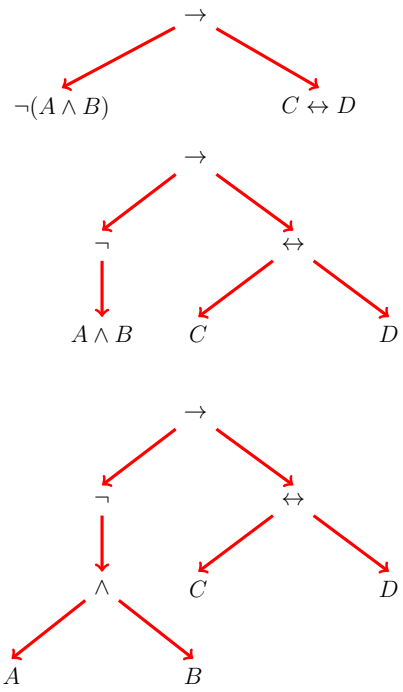


FIGURE 3.2 – Exemple de l'arbre de décomposition

Transformation en FNN

Pour transformer la formule en FNN, nous avons besoin de deux étapes.

1. Remplacer $\rightarrow$ et $\leftrightarrow$ par leurs connecteurs équivalent :
   $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
   $\varphi_1 \leftrightarrow \varphi_2 \equiv (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$.

   Remplacer $\rightarrow$: Afin de pouvoir remplacer $\rightarrow$ par $\neg\varphi_1 \vee \varphi_2$ dans l'arbre de décomposition, on met $\vee$ en tête d'arbre, et φ_1 à gauche de l'arbre et φ_2 à droite de l'arbre.

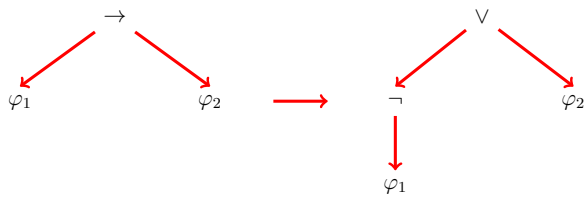


FIGURE 3.3 – Remplacement de l'implication dans l'arbre

Remplacer ↔ : Afin de pouvoir remplacer ↔ par $(\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$ dans l'arbre de décomposition, on met $\wedge$ en tête d'arbre, et $(\neg\varphi_1 \vee \varphi_2)$ à gauche de l'arbre et $(\varphi_1 \vee \neg\varphi_2)$ à droite de l'arbre.

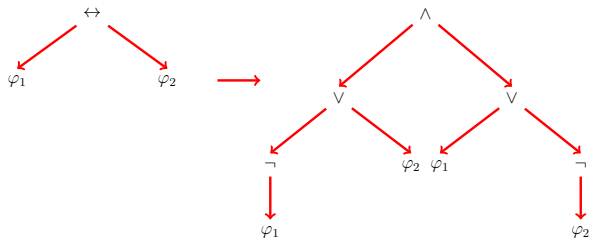


FIGURE 3.4 – Remplacement de l'équivalence dans l'arbre

2. Distribution de la négation : Il faut distribuer la négation sur la disjonction et la conjonction
   Notez que la formule se compose seulement de $\neg, \wedge, \vee$.
   $\neg(\varphi_1 \wedge \varphi_2) \equiv (\neg\varphi_1 \vee \neg\varphi_2)$.
   $\neg(\varphi_1 \vee \varphi_2) \equiv (\neg\varphi_1 \wedge \neg\varphi_2)$.
   $\neg(\neg\varphi) \equiv \varphi$.

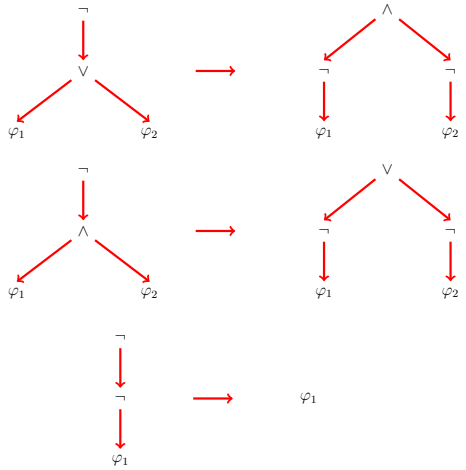


FIGURE 3.5 – Distribution de la négation dans l'arbre

Exemple : On applique ces deux étapes à l'arbre précédent, voir figure 3.2
$\varphi = (\neg(A \wedge B) \rightarrow (C \leftrightarrow D))$

la figure 3.6 présente l'arbre sous FNN.

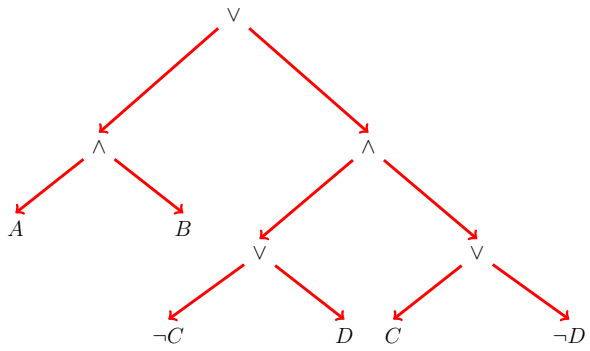


FIGURE 3.6 – Transformation FNN par l'arbre

$\text{FNN}(\varphi) = ((A \wedge B) \vee ((\neg C \vee D) \wedge (C \vee \neg D)))$

Transformation en FNC

Nous avons vu précédemment comment obtenir la FNC à partir de la table de vérité, mais la méthode de résolution a pour but de gagner la temps, nous avons suggéré une autre méthode qui dépend de FNN puis la distribuent la disjonction sur la conjonction.

Si la tête de l'arbre est $\vee$ et la tête de la sous-arbre gauche est $\wedge$ on distribue la sous-arbre gauche sur la sous-arbre droite. Sinon si la tête de la sous-arbre droite est $\wedge$ on distribue la sous-arbre droite sur la sous-arbre gauche . Et on applique ce principe à tous les nœuds de l'arbre.
notre idée est d'utiliser toujours l'arbre décomposition pour la distribuent.

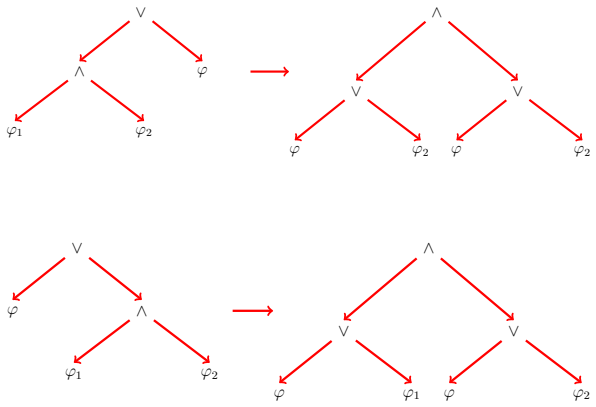


FIGURE 3.7 – Distribution $\vee$ sur $\wedge$

Exemple : On applique le principe de transformation en FNC à l'arbre de décomposition FNN de φ la figure3.6.
$\varphi = (\neg(A \wedge B) \rightarrow (C \leftrightarrow D))$
FNN(φ) est $((A \wedge B) \vee ((\neg C \vee D) \wedge (C \vee \neg D)))$

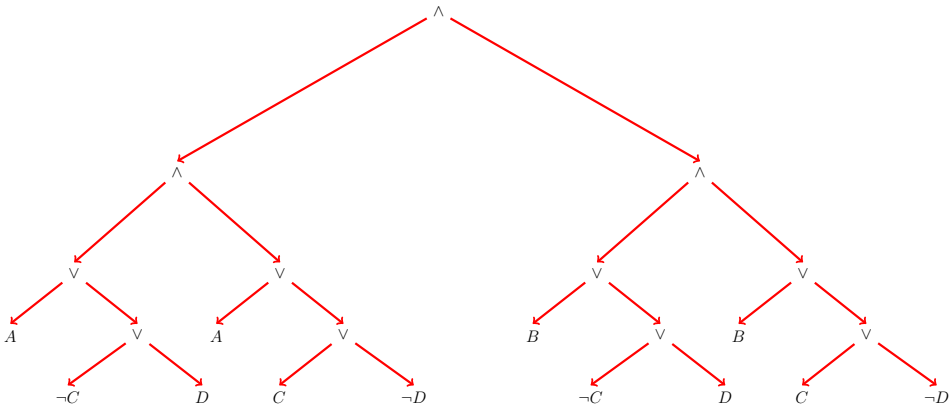


FIGURE 3.8 – Transformation FNC par l'arbre

Cet arbre dans la figure 3.8 est FNC.
FNC de $\varphi = ((A \vee \neg C \vee D) \wedge (A \vee C \vee \neg D) \wedge (B \vee \neg C \vee D) \wedge (B \vee C \vee \neg D))$

Extraction des clauses

Après avoir écrit la formule en forme FNC, nous pouvons former un ensemble de clauses , en extrayant les clauses de FNC
Exemple : $\varphi = (\neg(A \wedge B) \rightarrow (C \leftrightarrow D))$
FNC de $\varphi = ((A \vee \neg C \vee D) \wedge (A \vee C \vee \neg D) \wedge (B \vee \neg C \vee D) \wedge (B \vee C \vee \neg D))$
Σ est ensemble de clauses $\Sigma = \{A \vee \neg C \vee D, A \vee C \vee \neg D, B \vee \neg C \vee D, B \vee C \vee \neg D\}$

3.4.2 Notre système et la sélection des clauses

Après avoir expliquer comment Notre système convertir une formule à ses formes normals FNN et FNC puis extrait ensemble de clauses qui est utilisé dans la méthode résolution. Dans cette partie nous allons essayer de présenter nos interfaces de la résolution avec l'application des quelques exemples.

Choix aléatoire et aléatoire optimisé

Nous avons dit dans la section précédente que la méthode de résolution est compter sur la résolvante de deux clauses C1 et C2. Nous avons proposé deux techniques pour sélectionner les clauses pour résolution ,le choix aléatoire et résolution aléatoire optimisé.

1. Résolution avec le choix aléatoire : Cette technique garantie le choix aléatoire des clauses sans répétition C_i et C_j, en s'arrêtant jusqu'à atteindre $\perp$, si possible.
   Exemple : $\Sigma = \{A \vee \neg C \vee B, \neg A \vee C \vee B, A \vee C, \neg B \vee C, B, \neg C\}$

1	$A \vee \neg C \vee B$	Hypothèse
2	$\neg A \vee C \vee B$	Hypothèse
3	$A \vee C$	Hypothèse
4	$\neg B \vee C$	Hypothèse
5	B	Hypothèse
6	$\neg C$	Hypothèse
7	$\neg A \vee C$	Résolvant de 2,4
8	$A \vee B$	Résolvant de 3,1
9	$\neg A \vee B$	Résolvant de 6,2
10	$B \vee C$	Résolvant de 3,9
11	$C \vee \neg C \vee B$	Résolvant de 1,2
12	$\neg B$	Résolvant de 4,6
13	$A \vee \neg C$	Résolvant de 1,12
14	$\perp$	Résolvant de 5,12

2. Résolution avec le choix aléatoire optimisé : même principe que le choix aléatoire pour la sélection C_i et C_j mais nous avons ajouté deux conditions afin de minimiser l'ensemble des résolution :
   (a) Si $C_x, C_i \in \Sigma$, $\forall(C_x \subset C_i$, on ignore $C_i)$, car C_x nous donnera inévitablement le résultat avant C_i
   (b) Si C_x est tautologie, on l'ignore

   Exemple : $\Sigma = \{A \vee \neg C \vee B, \neg A \vee C \vee B \vee \neg A, A \vee C, \neg B \vee C, B, \neg C\}$

1	$A \vee C$	Hypothèse
2	$\neg B \vee C$	Hypothèse
3	B	Hypothèse (ignorer$A \vee \neg C \vee B$ et $\neg A \vee C \vee B \vee \neg A$)
4	$\neg C$	Hypothèse (ignorer $A \vee \neg C \vee B$)
5	A	Résolvant de 4,1 (ignorer 1)
6	$\neg B$	Résolvant de 4,2 (fait ignorer 2)
7	$\perp$	Résolvant de 6,3

Nous verrons plus tard des exemples de ces deux méthodes appliquées aux propriétés de la logiques.

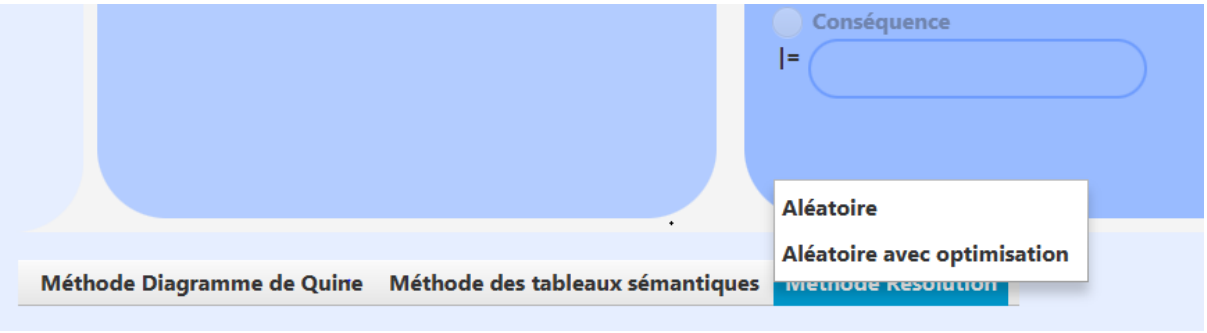


FIGURE 3.9 – Choix aléatoire et aléatoire optimisé

Discussion

Nous avons vu que la sélection des clauses est faite aléatoire donc le nombre des clauses résolvantes peut aller jusqu'à 2^n, N : est le nombre de littéraux dans Σ Si N est très grand alors notre application prend un temps important afin de donner un résultat.
Exemple $\Sigma = \{A \vee \neg C \vee D, \neg D \vee C \vee B \vee \neg A, A \vee C, \neg B \vee C, B, \neg A\}$
N $= 8 \Rightarrow$ nombre résolvants est environ $2^8 = 256$.

Afin de surmonter ce probleme, nous avons affre un champ pour déterminer le nombre d'itération voulu. Notre but est de ne pas perdre le temps sur un ensemble qui ne donne aucune clause vide.
Attention, dans le cas où le processus s'arrête à cause du nombre spécifié et il n'est atteindre le nombre d'itération possible, le résultat peut ne pas être exact.

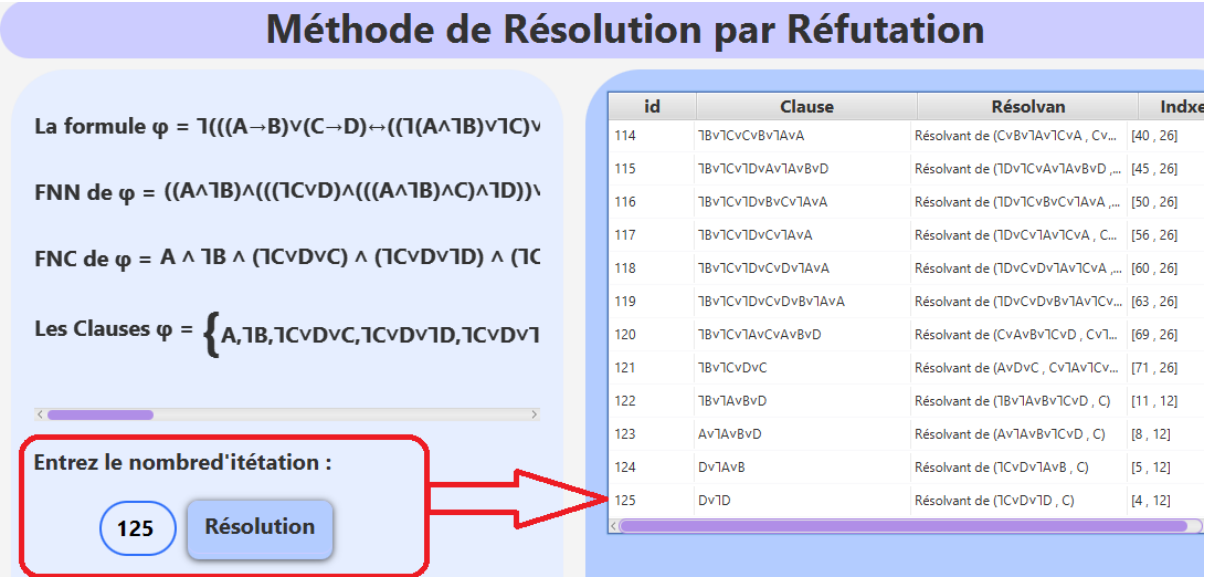


FIGURE 3.10 – Nombre d'itération voulu

Propriété satisfiabilité

Afin de montrer la satisfiabilité d'une formule φ en utilisant la méthode de résolution par réfutation, on prouve la déduction non vide à partir de l'ensemble des clauses de la formule φ. Une fois l'utilisateur taper une formule φ, sélectionne la propriété satisfiabilité , puis choisit la méthode résolution, l'application effectue plusieurs étapes sur la formule φ.

1. Transformation de φ en FNN.
2. Transformation FNN de φ en FNC.
3. Extraction des clauses à partir de FNC de φ dans Σ.
4. L'application demande à l'utilisateur de saisir le nombre voulu d'itérations pour la processus de résolution.
5. Enfin, l'utilisateur choisit la technique de la résolution (aléatoire et aléatoire avec optimisation).

Si on trouve $\perp$, alors la formule φ est insatisfiable, sinon la formule φ est satisfiable.
Exemple : Montrer que φ est satisfiable telle que
$\varphi = ((A \leftrightarrow (B \leftrightarrow (C \leftrightarrow D))) \leftrightarrow ((A \rightarrow B) \wedge (C \rightarrow D)))$.

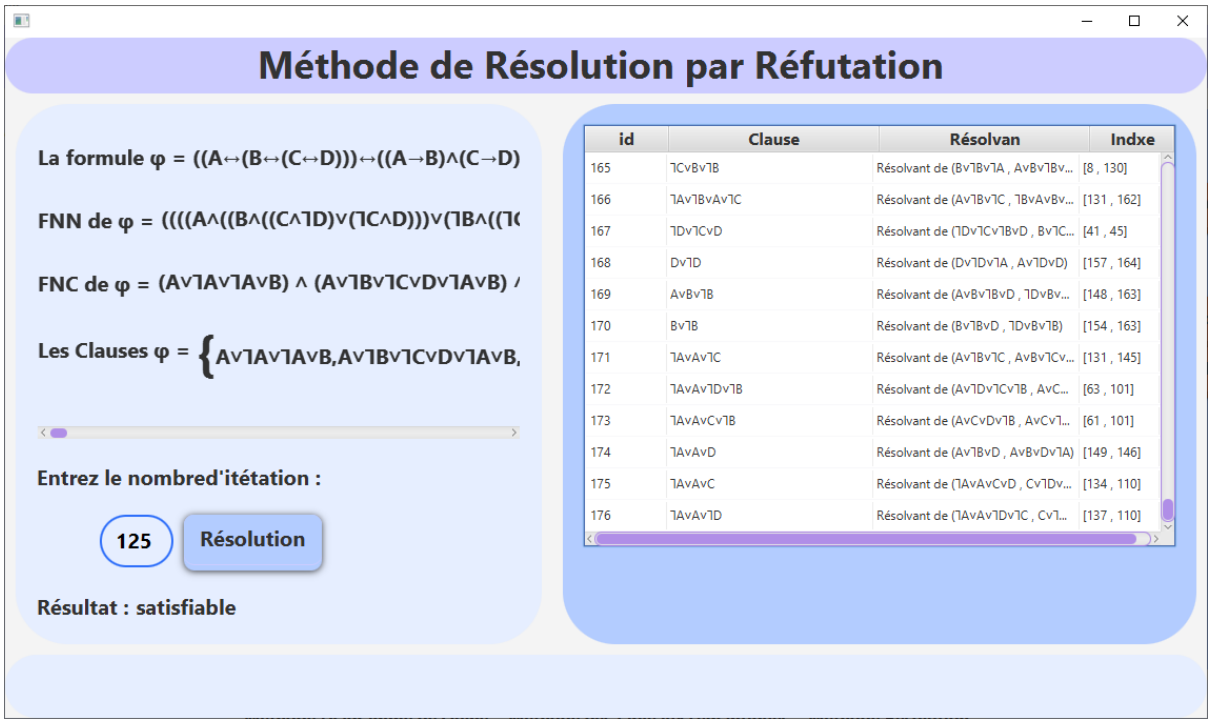


FIGURE 3.11 – Exemple de satisfiabilité avec résolution aléatoire

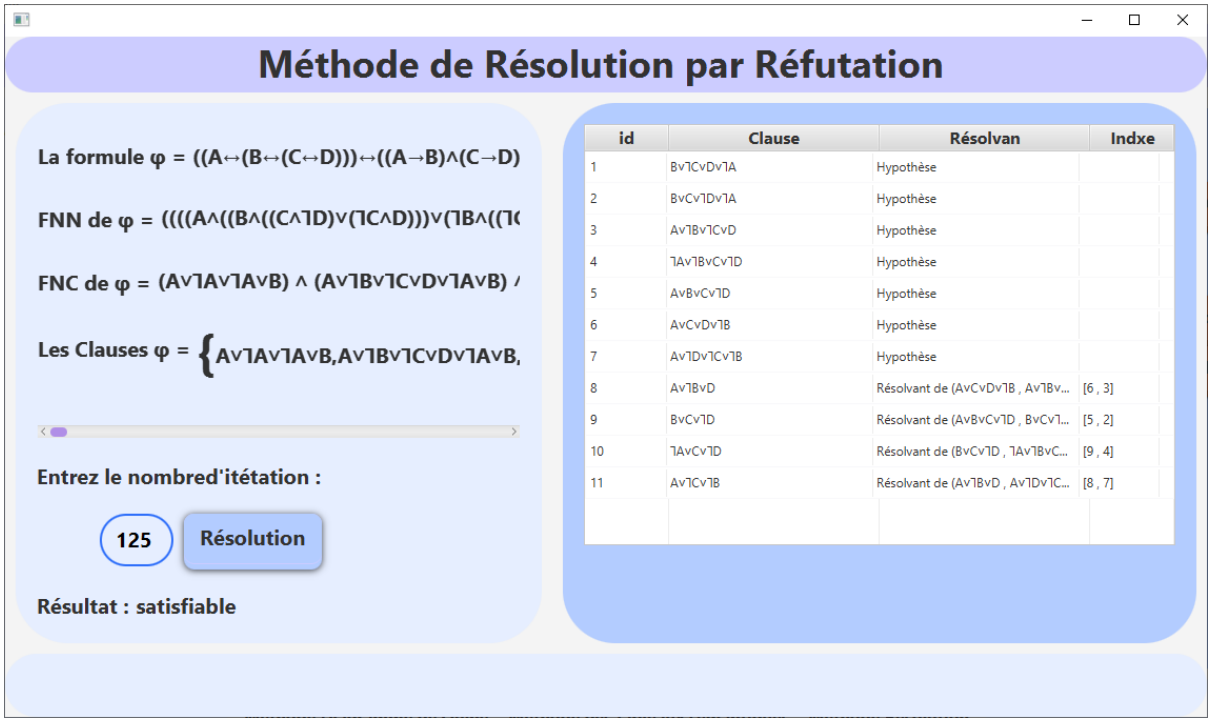


FIGURE 3.12 – Exemple de satisfiabilité avec résolution aléatoire optimisé

Propriété tautologie

Afin de montrer la tautologie (validité) d'une formule φ, on prouve la déduction de l'ensemble vide à partir de l'ensemble des clauses de sa négation (la négation de formule φ initiale). Une fois que l'utilisateur a tapé la formule φ, et a sélectionné la propriété tautologie, puis a choisi la méthode résolution. L'application traité automatiquement la négation de la formule et à effectuer des étapes suivantes sur $\neg\varphi$ comme suit

1. La négation est ajoutée à la formule φ.
2. Transformation $\neg\varphi$ en FNN.
3. Transformation FNN de $\neg\varphi$ en FNC.
4. Extraction des clauses à partir de FNC $\neg\varphi$ dans Σ.
5. L'application demande à l'utilisateur de saisir le nombre d'itérations des régles de résolution. Enfin, elle applique la méthode de résolution avec le choix aléatoire ou aléatoire optimisé à Σ.

Si on trouve $\perp$, alors la formule $\neg\varphi$ est antilogie $\Rightarrow$ φ est tautologie. Sinon φ est non valide. **Exemple :** Montrer que φ est tautologie telle que $\varphi = ((A \to B) \vee (C \to D) \leftrightarrow ((\neg(A \wedge \neg B) \vee \neg C) \vee D))$

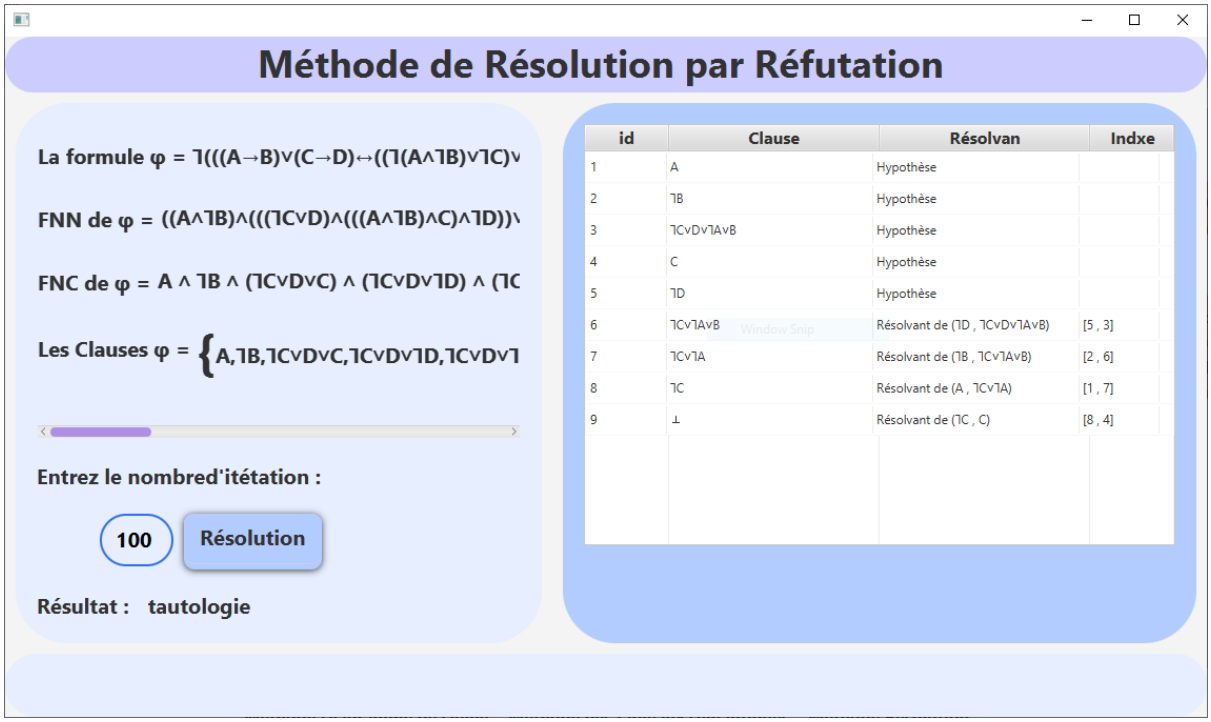


FIGURE 3.13 – Exemple de tautologie avec résolution aléatoire optimisé

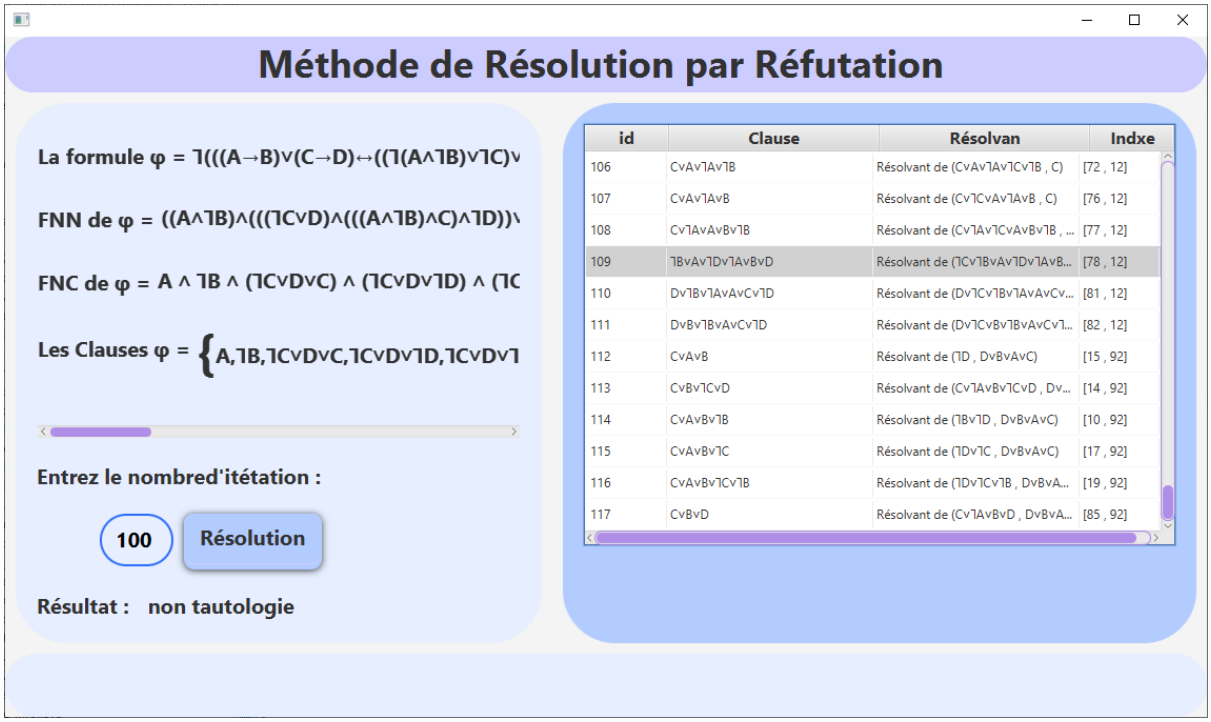


FIGURE 3.14 – Exemple de tautologie avec résolution aléatoire

Propriété antilogie

Afin de montrer l'antilogie (insatisfiable) d'une formule φ on prouve la déduction de l'ensemble vide à partir de l'ensemble des clauses de la formule φ initiale . Une fois

l'utilisateur a tapé la formule φ, et a sélectionné la propriété antilogie , puis a choisi la méthode résolution. L'application effectue les mêmes étapes de la propriété satisfiabilité avec un résultat différent. Si on trouve $\perp$, alors la formule φ est antilogie , sinon la formule φ est non satisfiable .

Exemple : Montrer que φ est antilogie telle que
$\varphi = ((\neg(A \rightarrow \neg B) \wedge ((C \vee D) \vee B)) \wedge ((A \wedge C) \wedge \neg(D \vee B)))$

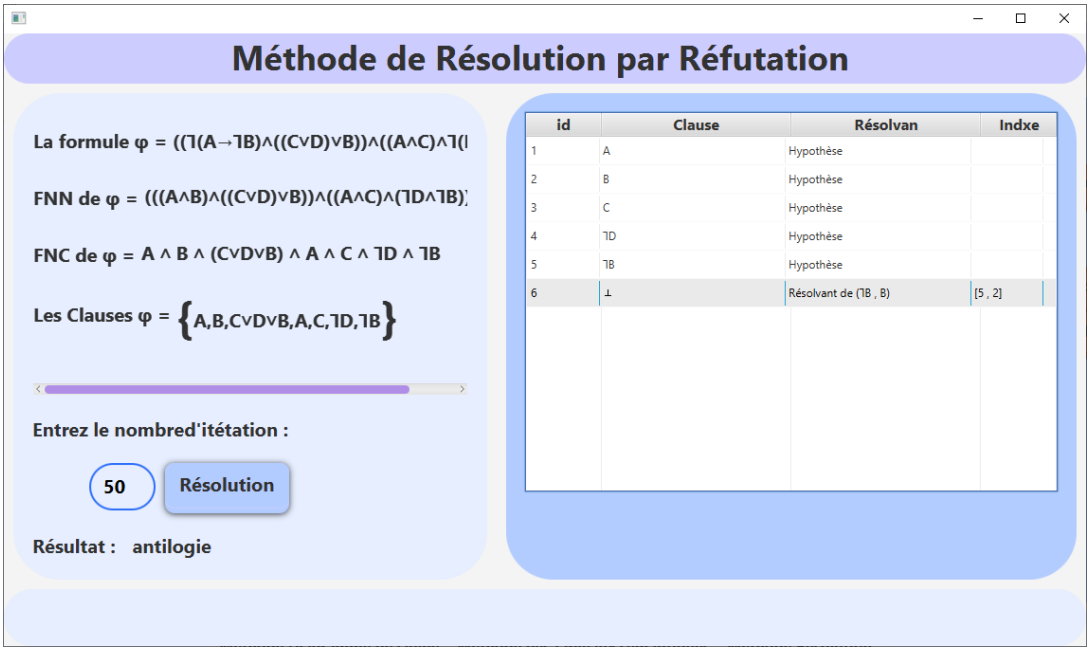


FIGURE 3.15 – Exemple de l'antilogie avec résolution aléatoire optimisé

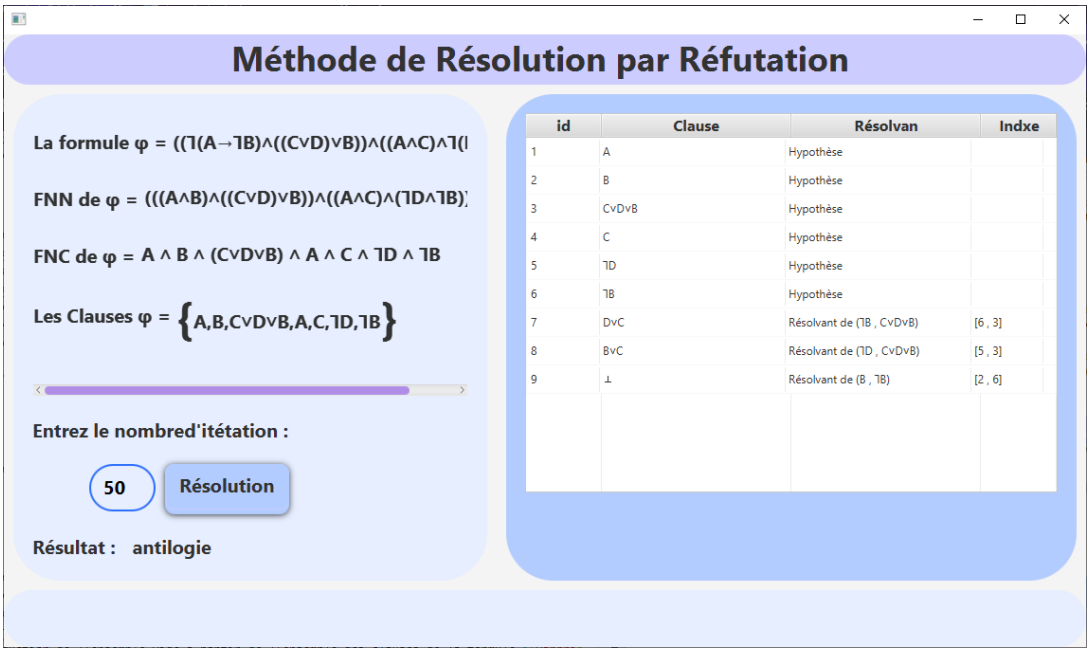


FIGURE 3.16 – Exemple de l'antilogie avec résolution aléatoire

Propriété de la compatibilité d'un ensemble de formule

Notre application fournit une option pour ajouter un ensemble de formules $\Sigma = \{\varphi_1, \varphi_2, \varphi_3, ..., \varphi_n\}$. Pour prouver la compatibilité de Σ, il faut étudier la satisfaction de ψ pour que $\psi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge ... \wedge \varphi_n$. L'application effectue les mêmes étapes que la propriété de satisfiabilité.
Si on trouve $\bot$, alors l'ensemble Σ est incompatible, sinon l'ensemble Σ est compatible.
Exemple : Montrer que Σ est compatible telle que $\Sigma = \{A \to (B \vee C), (B \wedge C) \to D, C \leftrightarrow D\}$

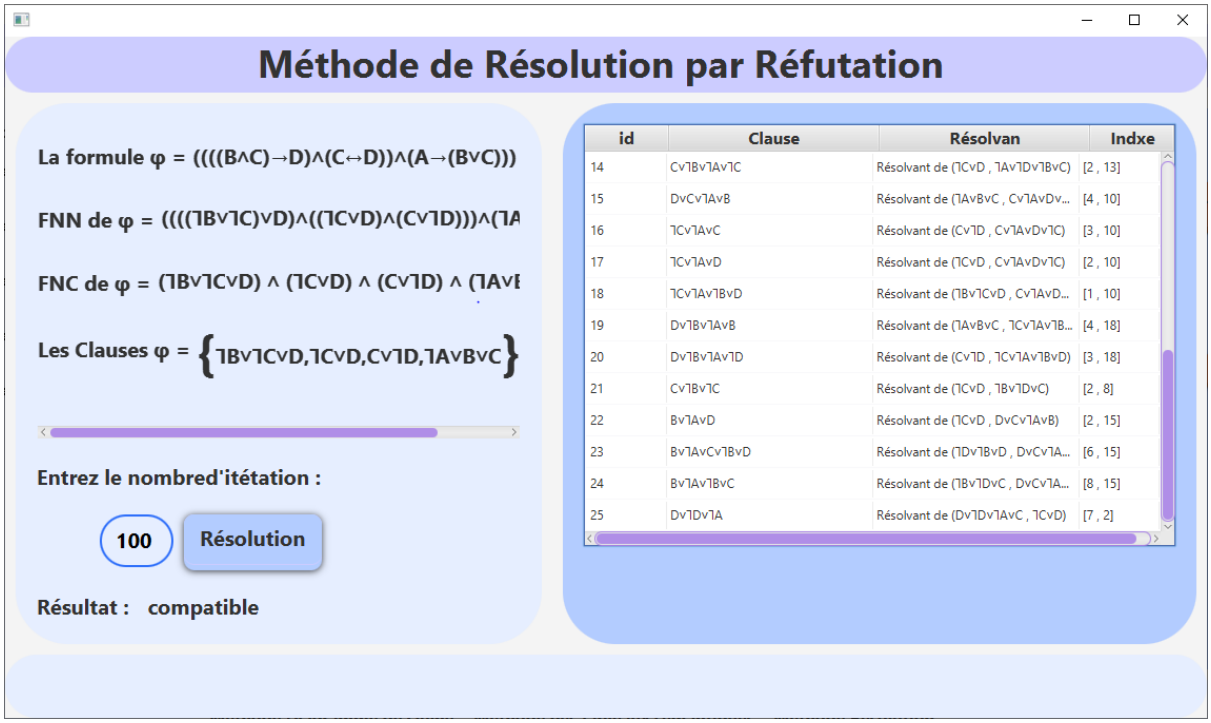


FIGURE 3.17 – Exemple de compatibilité avec résolution aléatoire

Propriété de la conséquence

Afin d'expliquer la preuve de conséquence logique $\Gamma \models \psi$ à l'aide de notre application telle que Γ un ensemble de formules $\Gamma = \{\psi_1, \psi_2, \psi_3, ..., \psi_n\}$ et ψ une formule. On a :
$\varphi = \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n \models \psi$
$\Rightarrow V_{((\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n) \to \psi)} = 1$
$\Rightarrow V_{(\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n \wedge \neg\psi)} = 0$
$\Rightarrow \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n \wedge \neg\psi \models \bot$
donc pour étudier la conséquence il revient d'étudier l'antilogie de $\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge ... \wedge \psi_n \wedge \neg\psi$.
Si l'application donne le résaltat $\bot \to \Gamma \models \psi$.
Sinon la conséquence n'existe pas.
Exemple : Montrer que $\Gamma \models \psi$ telle que $\Gamma = \{\neg(a \to b), (a \wedge (b \vee c)), (a \leftrightarrow b)\}$

est $\psi = a$

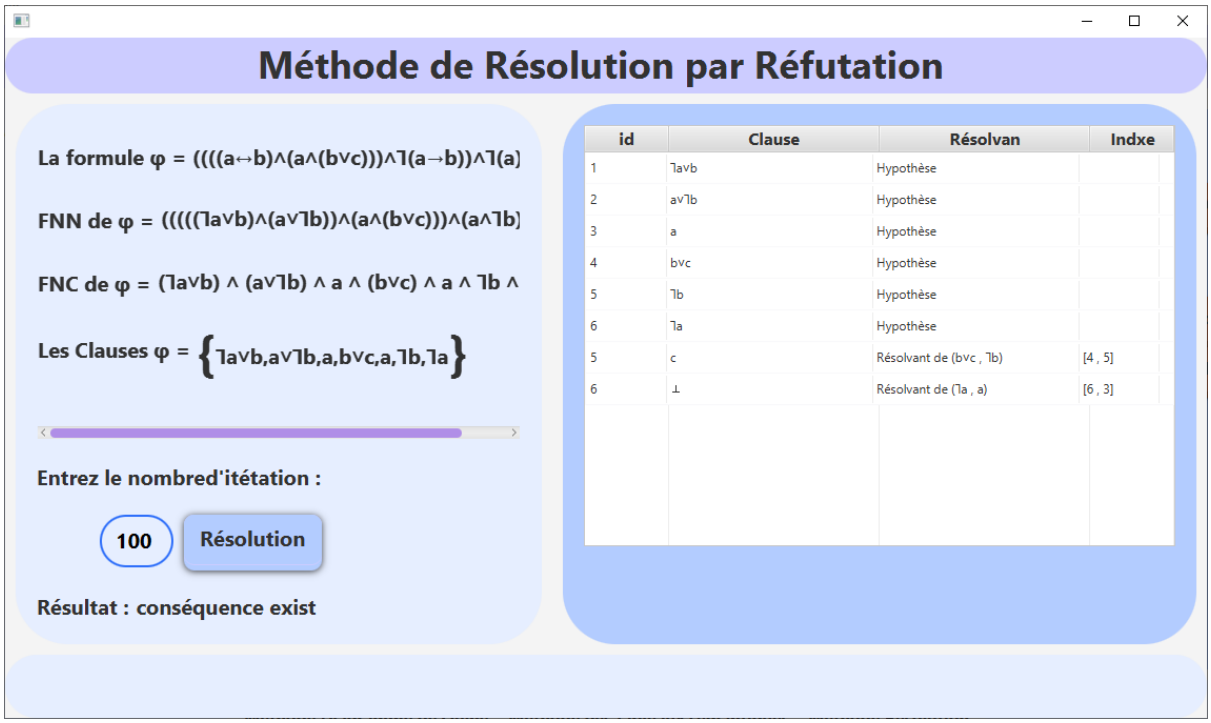


FIGURE 3.18 – Exemple de conséquence avec résolution aléatoire optimisé

3.5 Conclusion

Ce chapitre est divisé en trois parties. Nous avons commencé par la définition de toutes les formes normales d'une formule propositionnelle et comment les extraire à partir d'une table de vérité. Ensuite, nous avons décrit la méthode de résolution par réfutation en présentant leur principe, leur algorithme d'application et leur implémentation.

Conclusion Générale

La logique mathématique est l'une des sciences qui a utilisé par plusieurs domaines, nous nous somme intéressés par le type propositionnel qu'est le plus simple et le plus utilisé.

Plus précisément, nous avons étudié quels que systèmes de preuve, le diagramme de Quine qui simplifie largement les interprétations possibles des formules propositionnelles, la méthode des tableaux sémantiques qui consiste en la recherche systématique d'un modèle d'une formule donnée, ou évidemment un anti-modèle et enfin la méthode de résolution par réfutation qui permet de démontrer la validité d'une formule en démontrant l'inconsistance de sa négation.

Face aux difficultés rencontrées dans l'application pratique de ces méthodes, nous avons proposé de les automatiser à l'aide de l'outil informatique. L'objectif de notre projet de fin d'étude était d'implémenter et de développer un environnement de raisonnement propositionnel basé sur ces trois méthodes de preuve. Notre proposition vise à simplifier les tâches de raisonnement logique en donnant des résultats aux problèmes logiques complexes dans les plus brefs délais d'une manière simple et très facile. Notre outil présenté gagne beaucoup de puissance du fait qu'il repose sur des concepts formels de la logique propositionnelle et son système de preuve.

A travers notre outil, nous avons appliqué ces techniques à n'importe quel ensemble de formules afin de prouver les propriétés logiques, la validité, la satisfiabilité, la compatibilité et même la conséquence. Notre environnement crée la simplicité, l'efficacité et l'exactitude des résultats en améliorant les fonctionnalités des méthodes sélectionnées par les choix offerts.

Notre travail développé constitue une étape qui pourrait contribuer au bénéfice des étudiants, professeurs et chercheurs spécialisés dans les domaines mathématique et informatique en réduisant les efforts et le temps qu'ils y consacrent.

Ce projet nous couté beaucoup de temps mais vraiment c'est une expérience très intéressante. Elle nous donne la main de manipuler des programmes dans notre domaine d'étude, également nous avons appris largement sur la programmation logicielle et la conception d'application.

Malgré les efforts que nous avons déployés pour accomplir cet humble travail et malgré les perceptions que nous avons acquises, nous pensons que notre contribution n'est que le début d'un long chemin. Le travail que nous avons effectué pourrait être amélioré, complété et suivi de plusieurs façons, notamment :

— Utiliser l'algorithme des tableaux sémantique de manière optimale, en d'autres termes arrêter le processus de calcul dès que on trouve un modèle (le cas de la satisfiabilité) ou si on trouve une feuille fermée qui comporte deux littéraux complémentaires (le cas de la validité).

— Faire une étude comparative entre les trois méthodes de preuve.

— Généralisé l'utilisation de ces méthodes par d'autres types de logique.

— Etudier d'autres méthodes de preuve.

Bibliographie

[1] Logique propositionnelle. URL https://studylibfr.com/doc/528243/7-logique-propositionn

[2] 09/06/2022. URL https://www.latex-project.org/.

[3] 09/06/2022. URL https://www.eclipse.org/downloads/.

[4] 09/06/2022. URL https://www.java.com/fr.

[5] 09/06/2022. URL https://openjfx.io/.

[6] 09/06/2022. URL https://www.oracle.com/java/technologies/downloads/.

[7] D. ADEL and A. Karima. Logique mathématique. 2015-2016.

[8] L. BELKACEMI. Formalisation de la logique temporelle dans coq. 2013.

[9] Broché. Livre methodes de logique (français). 1 janvier 1991.

[10] R. Cori and D. Lascar. Logique mathématique cours et exercices tome 1 calcul propositionnel, algébres de boole, calcul des prédicats. axiomes. 1993.

[11] M. Cozic. logique propositionnelle, 2 sémantique de lp, logique séance 3. 2012-2013.

[12] S. Devismes, P. Lafourcade, and M. Lévy. *Logique et démonstration automatique : introduction à la logique propositionnelle et à la logique du premier ordre*. Ellipses, 2012.

[13] E. Filiot. Logique pour l'informatique. 2011.

[14] P. Gribomont. Logique,université de liège. 2006.

[15] E. Mendelson. Introduction to mathematical logic. 1964.

[16] D. Pastre. Logique et principe de résolution.

[17] J. Stern. Fondements mathématiques de l'informatique. 1990.