République Algérienne Démocratique
et Populaire
Université Abou Bakr Belkaid
Tlemcen
Faculté des Sciences
Département d'Informatique

Intitulé du Polycopié:

Introduction à la Logique Mathématique

Dr. Hadjila Fethallah

fethallah.hadjila@univ-tlemcen.dz

2 avril, 2019

Préambule

Ce polycopié est destiné aux étudiants de la deuxième année licence en informatique. Il présente les notions de base de la la logique mathématique et la calculabilité et les consolide avec des exemples. Il est aussi destiné aux étudiants en master désirant approfondir leurs connaissances dans le domaine des solveurs SAT ou du raisonnement formel.

Dédicaces

A toute ma famille, et mon cher ami Amine.

Préface

La logique mathématique constitue l'une des disciplines majeures qui ont fondé l'informatique actuelle, selon [24], la logique mathématique est un domaine qui étudie les méthodes de raisonnement valide. En d'autres mots, elle nous offre des modèles, des concepts et des algorithmes pour calculer la valeur de vérité des formules logiques ainsi que de prouver l'existence de relations de deductibilité (conséquence) entre formules.

L'histoire récente de la logique mathématique a commencé vers la fin du 19ème siècle. Durant cette période, il y avait des tentatives ambitieuses pour donner une base solide à la théorie des ensembles. Plus précisément, le mathématicien autrichien **Cantor** a représenté la théorie des ensembles à l'aide d'une collection de principes informels (ou axiomes). Mais cette axiomatisation était trop limitée et même inconsistante, puisqu'un certain nombre de chercheurs ont découvert beaucoup d'erreurs/paradoxes à cette approche. Nous citons à titre d'exemples 02 paradoxes (le lecteur intéréssé trouvera plus d'informations sur les paradoxes et leurs types dans la reférence [29]):
Paradoxe de Russell(1902): "soit r l'ensemble des ensembles ne contenant pas eux mêmes". Après une simple analyse, il s'avère que R n'existe pas, parce-que cette définition de R implique la propriété suivante:
l'ensemble R appartient à R si et seulement si R n'appartient pas à R, malheureusement la théorie de Cantor ne peut pas éliminer ce genre de paradoxe.
Paradoxe de l'ensemble contenant tous les ensembles: (Cantor 1899)
"soit U l'ensemble de tous les ensembles".
A partir de cette définition, nous constatons que U contient U comme membre (puisque U est un ensemble) et meme l'ensemble des parties de U (2^U) puisque ce dernier est aussi un ensemble, et meme tous les elements de 2^U. A partir de cela, nous constatons que la taille de U est plus grande ou égale que $|2^U|$, cad que $|U| \geq |2^U|$. Mais Cantor a deja demontré dans le passé que la taille de n'importe quel ensemble S est strictement inferieure à celle de 2^S. (et entre autres $|U| < |2^U|$). Donc on a une contradiction concernant la taille de U (qui ne peut exister), et par consequent l'unique solution à ce paradoxe est que U ne peut exister. Malheureusement cela aussi implique que les axiomes infomels de Cantor sont erronés (parce qui'ils ont pu definir un ensemble comme U).

Cette situation de crise de fondement a poussé des chercheurs comme Russel, Zermelo et Hilbert à créer des théories exemptes d'erreurs. Par exemple, Zermelo et d'autres chercheurs ont crée une axiomatisation de la théorie naive des ensembles de Cantor notée ZFC (Théorie de Zermelo-Fraenkel avec axiome du Choix). De même, Hilbert a creé un programme ambitieux dont le but est de prouver la cohérence des mathématiques à l'aide des systèmes axiomatiques (i.e., un langage formel contenant des axiomes et des regles d'inferences). En

particulier, il a énoncé une liste 23 problèmes à résoudre et dont le deuxième concernait la cohérence de l'arithmétique (est ce qu-on peut démontrer que les axiomes de l'arithmétique -i.e., les entires naturels et leurs propriétés- ne sont pas contradictoires, i.e., ils ne peuvent pas dériver une formule F et sa négation en même temps ?). Hilbert voulait prouver la cohérence des mathématiques, mais puisque ce vaste domaine était trés complexe, il se limitera avec ses collègues d'axiomatiser uniquement l'arithmétique (qui était considérée comme le noyau des mathématiques). Quelques années plutard, un jeune mathématicien nommé Godel a démontré en 1931 (premier théorème d'incomplétude) que la preuve de cohérence de l'arithmétique dans le système axiomatique de l'arithmétique est impossible. En particulier, Godel a énoncé deux théorèmes, le premier affirme que toute théorie axiomatisable (et consistante) couvrant l'arithmétique est incomplete (i.e., il y a des formules valides dans la théorie qui n'ont pas de preuve). Le deuxième théorème stipule que l'arithmétique elle même ne peut pas démontrer sa consistante (ou cohérence). Ceci a donné une fin un peu tragique aux travaux d'Hilbert, puisque la démonstration de la cohérence des mathématique s'avére nonfaisable (indécidable). En parallèle, un ensemble de chercheurs tels que Turing et Church ont travaillé sur la notion de calculabilité et décidabilité des problèmes (et effictivement dire s'il y a une preuve pour la véracité ou la fausseté d'une proposition quelquonque P en mathématique); par exemple, dans un système axiomatique donné (celui de l'arithmétique, ZFC, ou autre), on veut avoir un algorithme qui permet de dire si une formule F sur les ensembles est prouvable ou non, cette procédure algorithmique doit s'arréter pour toute entrée possible (et dans ce cas, le problème est dit décidable). Pour réaliser cet objectif, il faut représenter mathématiquement la notion d'algorithme, et dans cette optique, le chercheur britanique Turing a inventé en 1936 une machine théorique (nommée machine de Turing) qui permet de reconnaitre si un problème est décidable ou non (tel que la déductiblité dans un système axiomatique donné). Bien que cette machine n'est pas réalisable sur le terrain (puisqu'elle requiert une mémoire infinie), elle permet de montrer que certains problèmes ne peuvent jamais être solutionnés sur un ordianteur (problèmes indécidables).

De façon générale, nous pouvons distinguer quatre branches de la logique mathématique :

- La théorie des ensembles.
- La théorie des modèles.
- La théorie de la preuve.
- La calculabilité.

La théorie des ensembles s'intéresse à l'étude des ensembles ainsi que leurs propriétés (ex: la dénombrabilité, la cardinalité, etc.)
La théorie des modèles offre des concepts (structures, règles, fonctions) pour donner une signification aux formules logiques ainsi que leurs constituants. Grâce à ces interprétations, nous pouvons créer une première approche pour raisonner sur les formules.
La théorie de la preuve constitue une approche alternative qui permet de raisonner sur les formules logiques sans utiliser les interprétations (la sémantique

des formules). Grosso-modo, la théorie de la preuve conçoit les démonstrations comme une suite finie de formules logiques. Chacune d'elles est soit un axiome, soit une application d'une règle d'inférence sur d'autres formules (déjà démontrées) ou axiomes.
La calculabilité est une théorie qui a pour objectif de formaliser la notion de procédure effective (les algorithmes dont l'exécution s'arrête toujours), pour ce faire, elle étudie plusieurs modèles de calcul tels que: la machine de Turing, le Lamda-Calcul, les fonctions récursives, etc.

Grâce à cette discipline, nous pouvons aussi distinguer entre un problème calculable (réalisable sur la machine) et un problème non calculable (non automatisable sur une machine ou n'ayant pas de procédure effective). Nous pouvons aussi classer problèmes selon leur degré de difficulté; par exemple, nous pouvons faire une distinction entre les problèmes ayant des solutions efficaces ou polynomiales (ex: le tri d'un tableau, le produit scalaire de deux vecteurs, etc.), et les problèmes n'ayant pas de solutions efficaces (ex: la résolution d'un jeu d'échec, la factorisation des grand nombres entiers, etc.)

Dans ce modeste polycopié, nous couvrerons la théorie des modèles et la théorie de la preuve (dans les chapitres 1 et 2), nous introduisons aussi la calculabilité en se focalisant sur le modèle de la machine de Turing (voir le chapitre 3).
.

Table des Matières

Liste des figures

Chapitre 1

Introduction à la logique des propositions

1.1 Introduction

Ce chapitre présente le langage logique le plus simple appelé aussi logique propositionnelle ou logique d'ordre 0. Il est ainsi qualifié parce que ce langage ne possède pas de variables manipulées au niveau des propositions. L'alphabet de cette logique est constitué de plusieurs groupes de symboles, dont le plus important est celui des propositions. Informellement une proposition est un énonce déclaratif qui peut être vrai ou faux, nous montrons ci-dessous quelques exemples de propositions:

1. Tlemcen est la capitale des Zianides
2. Tous les algériens sont riches.
3. La terre est ronde.

Par opposition, les exemples suivants ne sont pas des propositions (soit parcequ'elles ne sont pas déclaratives, ou on peut pas leurs associer une valeur de vérité 1 ou 0).

1. Sortez maintenant ! (Expression de impérative).
2. Quelle heure est il ? (Expression interrogative).
3. Quelle belle photo! (Expression exclamative).
4. Il est présent. (Expression déclarative , mais elle ne peut avoir une valeur de vérité à cause de la présence de variables libres (i.e., il)).
5. x+2=y (Expression déclarative , mais elle ne peut avoir une valeur de vérité à cause de la présence de variables libres x et y).
6. Cette expression est fausse. (Expression déclarative, mais elle ne peut avoir une valeur de vérité à cause de la présence d'auto-référence).

Dans ce qui suit, nous présentons la syntaxe du langage, sa sémantique (théorie des modèles), son fameux système de preuve dénommé **"la résolution"**, nous aborderons aussi les problèmes de satis-faisabilité et les solutions offertes (Solveurs SAT).

1.2 Synatxe

Définition 1 *L'alphabet de la logique des propositions est défini comme suit:*

1. *un ensemble dénombrable P de variables propositionnelles (ou formules atomiques, ou encore atomes ou propositions simples).*
2. *La constante $\perp$ (qui représente le zero logique).*
3. *Les connecteurs $\rceil, \wedge, \vee, \rightarrow, \leftrightarrow$.*
4. *Les parenthèses (,).*

Nous utilisons les symboles $p_1, p_2, ..q_1, q_2, ..r_1, r_2, \ldots$, pour dénoter les variables propositionnelles.

Définition 2 *L'ensemble des formules bien formées (fbf) de la logique propositionnelle (noté aussi FOR) peut être défini par récurrence comme suit:*

- *Si A est une formule atomique alors A est une fbf.*
- *$\perp$ est une fbf.*
- *$\rceil A$ est une fbf si A est une fbf.*
- *$A \wedge B$ est une fbf si A et B sont des fbf.*
- *$A \vee B$ est une fbf si A et B sont des fbf.*
- *$A \leftrightarrow B$ est une fbf si A et B sont des fbf.*
- *$A \rightarrow B$ est une fbf si A et B sont des fbf.*

Illustration de fbf

$C \rightarrow (A \wedge B)$, $(A \wedge B) \leftrightarrow (\rceil A \wedge C), \cdots$

1.3 Théorie des modèles

1.3.1 Notion d'interprétation

L'objectif de la théorie des modèles est de donner un sens à chaque fbf. Le sens est une valeur appartenant à un ensemble D appelé domaine d'interprétation. pour la logique propositionnelle, D = $\{1, 0\}$ (vrai, faux).

Définition 3 *Une interprétation I est une application de l'ensemble des variables propositionnelles P dans l'ensemble des valeurs de vérité* $\{1, 0\}$. $I : P \rightarrow \{0, 1\}$
Chaque interprétation I peut être étendue à une fonction I' qui s'applique aux formules bien formées, comme suit :

1. $I'(\bot) = 0$.
2. $I'(A) = I(A)$ *pour tout* $A \in P$.
3. $I'(\neg A) = 1 - I'(A)$.
4. $I'(A \wedge B) = min(I'(A), I'(B))$.
5. $I'(A \vee B) = max(I'(A), I'(B))$.
6. *si (I'(A) = 1 et I'(B) = 0) alors 0, sinon 1.*
7. *si (I'(A) = I'(B)) alors 1, sinon 0.*

Remarques

- Informellement une interprétation I' (ou valuation ou monde possible) associée à une fbf H est une ligne de sa table de vérité.
- Pour une formule à n atomes, il y a 2^n mondes possibles.
- I' est un modèle de A (ou I' vérifie A) ssi I'(A) = 1.
- I' est un modèle pour un ensemble de formules S ssi I' est un modèle pour toute formule A de S.
- I' est un anti-modèle de A ssi I'(A) = 0.

La table 1.1 montre les interprétations possibles de la formule F: $(A \rightarrow B) \vee B$

Table 1.1: Table de vérité de F

A	**B**	**A→ B**	**F**
0	0	1	1
1	0	0	0
0	1	1	1
1	1	1	1

Nous constatons par exemple que la première ligne représente le modèle de F, alors que la deuxième est un anti-modèle de F.

1.3.2 Classification des formules propositionnelles

En général, les formules bien formées sont reparties en quatre types majeurs: les formules satisfaisables, insatisfaisables, valides et contingentes.

Définition 4 (Formule satisfaisable (consistante)) *C'est une formule qui possède au moins un modèle dans sa table de vérité (i.e., il existe une interprétation telle que la formule est vraie).*

Exemple 1.1.
$(A \rightarrow B), (A \vee B), \ldots$

Définition 5 (Formule insatisfaisable (inconsistante)) *C'est une formule qui ne possède que des anti-modèles dans sa table de vérité (i.e., toutes les interprétations rendent la formule fausse).*

Exemple 1.2.
$((A \rightarrow \rceil A) \wedge A), (A \wedge \rceil A), \ldots$

Définition 6 (Formule valide (tautologie)) *C'est une formule qui ne possède que des modèles dans sa table de vérité (i.e., toutes les interprétations rendent la formule vraie).*
Une formule valide A est notée $\models A$.

Exemple 1.3.
$(A \rightarrow A), (A \vee \rceil A), ((A \rightarrow B) \rightarrow (\rceil B \rightarrow \rceil A)) \ldots$

Définition 7 (Formule contingente) *C'est une formule qui possède au moins un modèle et au moins un anti-modèle dans sa table de vérité (i.e., il existe une interprétation pour la quelle la formule est vraie, et il existe une autre pour laquelle la formule est fausse).*

Exemple 1.4.
$(A \rightarrow B), (A \vee B) \rightarrow A$, etc.

Remarques

- La tâche de décider si une formule est satisfaisable ou non est connue sous l'appellation de **"problème SAT"**, les solutions de ce problème sont détaillées dans la section 1.7.
- Le problème SAT est NP-Complet (voir le chapitre 3 pour plus de détail)

1.3.3 Notion de conséquence sémantique (ou logique)

Définition 8 (Conséquence sémantique (ou logique)) *Soient Γ un ensemble de formules $\in FOR$ et A une formule $\in FOR$: A est une conséquence sémantique de l'ensemble de formules Γ si tout modèle de Γ est un modèle de A. Cette relation est notée comme suit: $\Gamma \models A$.*

Exemple 1

D'après la table de vérité suivante, la conséquence sémantique est vérifiée pour H1 et H3 ($H1 \models H3$), mais pas pour H1 et H2 (H2 n'est pas une conséquence sémantique de H1, puisque la troisième ligne de la table de vérité contredit la définition).

A	B	C	$B \leftrightarrow C$ (H1)	$\rceil A \rightarrow (B \wedge C)$(H2)	$(A \vee B \vee \rceil C))$(H3)
1	1	1	1	1	1
0	1	1	1	1	1
1	0	1	1	0	1
0	0	1	0	0	0
1	1	0	1	0	1
0	1	0	0	0	1
1	0	0	1	1	1
0	0	0	0	1	1

Définition 9 (Equivalence sémantique (ou logique)) *Soient A et B deux formules* $\in FOR$ *: A est logiquement équivalente à B ssi* $A \models B$ *et* $B \models A$.

Remarques

- $si A \models B$ alors$\models A \rightarrow B$
- Une formule valide A est une conséquence sémantique de l'ensemble vide : $\emptyset \models A$

1.4 Formes Normales

Pour faciliter la mise en œuvre des algorithmes de raisonnement, il sera utile de transformer une formule en une autre (qui est logiquement équivalente) et qui a une syntaxe plus simple (standardisée). Dans ce qui suit, nous introduisons deux formes normales: la forme normale conjonctive (FNC) et la forme normale disjonctive(FND). Notons que la forme normale conjonctive se base sur la notion de clause, alors que la FND se base sur la notion de monôme. En plus, la clause et le monôme se basent sur le concept de littéral.

Définition 10 (Littéral) *Un littéral est une formule logique de la forme A ou* $\rceil A$*, où A est une variable propositionnelle.*

Exemple 1.5.
$a, \rceil a$

Définition 11 (Clause) *Une clause est une disjonction de littéraux.*

Exemple 1.6.
$(\rceil a \vee b \vee c \vee d)$
Remarque
La clause vide notée aussi $(\square)$(le nombre de littéraux est égal à 0) représente le zero logique (i.e., $\perp$).

Définition 12 (Forme normale conjonctive) *La forme normale conjonctive (FNC) est une conjonction de clauses.*

la FNC permet de localiser facilement des contre-modèles.
Exemple 1.7.
$(\rceil a \vee b \vee c) \wedge (\rceil c \vee \rceil a \vee c) \wedge (a \vee \rceil b)$

Définition 13 (monôme) *Un monôme est une conjonction de littéraux.*

Exemple 1.8.
$(\rceil a \wedge b \wedge c \wedge \rceil d)$

Définition 14 (Forme normale disjonctive) *La forme normale disjonctive (FND) est une disjonction de monômes.*

La FND permet de localiser facilement les modèles.
Exemple 1.9.
$(a \wedge b) \vee (\rceil a \wedge \rceil b \wedge c)$

Théoreme 1 *Toute formule logique est équivalente à une disjonction de conjonctions de littéraux; de même, toute formule logique est équivalente à une conjonction de disjonctions de littéraux.*

Méthode de transformation en forme normale conjonctive (FNC)

Nous donnons par la suite une méthode (voir les règles citées ci-dessous) pour élaborer la forme normale conjonctive associée à une formule donnée.

1. Elimination des connecteurs $\leftrightarrow$ *et* $\rightarrow$:
   $F \leftrightarrow G$ devient $(F \rightarrow G) \wedge (G \rightarrow F)$, $(F \rightarrow G)$ devient $(\urcorner F \vee G)$.
2. Décalage des négations à l'intérieur de la formule : $\urcorner(F \wedge G)$ devient $(\urcorner F \vee \urcorner G)$, $\urcorner(F \vee G)$ devient $(\urcorner F \wedge \urcorner G)$.
3. Distributivité de $\vee$ par rapport à $\wedge$: $F \vee (G \wedge H))$ devient $((F \vee G) \wedge (F \vee H))$.

Exemple 2 *Mettre la formule* : $(p \rightarrow ((\urcorner q) \vee r)) \vee (t \wedge \urcorner q)$ *sous forme normale conjonctive.*

Après traitement, la formule devient:
$(\urcorner p \vee \urcorner q \vee r \vee \urcorner q) \wedge (\urcorner p \vee \urcorner q \vee r \vee t)$

1.5 Théorie de la preuve

1.5.1 Introduction

La théorie de la preuve est une alternative à la théorie des modèles qui permet d'établir des relations entre les formules logiques sans passer par la table de vérité (qui est généralement longue et coûteuse en termes de temps). Dans ce qui suit, nous présentons les ingerdients de base d'un système formel (ou système de preuve), ainsi que la notion de déduction et de preuve.

Définition 15 *selon (cameron, 98)un système formel (ou système de preuve, système déductif) est un quadruplet noté (AL,FO,AX,RI) avec:*

1. *AL: est un alphabet dont les éléments sont appelés symboles (dans le cas de la logique des propositions, ceci couvre les variables propositionnelles, les connecteurs, etc.).*
2. *FO: est un sous ensemble de AL^*, il est aussi appelé l'ensemble des formules, et il se définit grâce à des règles grammaticales.*
3. *AX: est un ensemble d'axiomes, un axiome est une formule valide qui appartient FO.*
4. *RI: est un ensemble fini de règles qui permettent de dériver les formules, les prémisses et les conclusions d'une règle d'inférence appartiennent à FO.*

Exemple 3 (Le système de Lukasiewicz) *Ce système est défini comme suit:*

1. *AL =$\{\urcorner, \rightarrow\} \cup P$*
2. *FO: toutes les formules logiques construites grâce aux connecteurs $\urcorner, \rightarrow$*
3. *AX : il est constitué des 03 formules suivantes:*

   (a) $A_1 : P \to (Q \to P)$
   (b) $A_2 : (P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$
   (c) $A_3 : (\neg P \to \neg Q) \to (Q \to P)$

4. *RI il est constitué des 02 règles suivantes:*
   (a) *Modus Ponens:* $P, (P \to Q) \vdash Q$.
   (b) *Règle de substitution: si* $A \in FO$, *alors nous dérivons B de A, en remplaçons uniformément les mêmes symboles de A par les mêmes formules (de substitution).*

Définition 16 (Déduction) *Soient (AL,FO,AX,RI) un système preuve et* $H_1, ..., H_n$,F *des formules appartenant à FO, une déduction de F à partir des hypothèses* $H_1, ..., H_n$ *est une suite de formules* $F_1, ..., F_k \in FO$ *où chaque* F_i *est soit :*

1. *Une hypothèse.*
2. *Un axiome.*
3. *Ou une formule obtenue à partir des règles inférence appliquées aux formules placées avant* F_i *(ie, les formules* $F_1, ..., F_{i-1}$*).*
4. F *est la derniere formule* F_k.

On note la déduction (ou dérivation) comme suit:$\{H_1, ..., H_n\} \vdash F$

Définition 17 (Preuve) *Une formule Q est prouvable dans le système preuve (AL,FO,AX,RI), s'il existe une suite de formules* $F_1, ..., F_n \in FO$, *où chaque* F_i *est soit :*

1. *Un axiome.*
2. *Ou une formule obtenue à partir des règles inférence appliquées aux formules placées avant* F_i *, (ie, les formules* $F_1, ..., F_{i-1}$*).*

Q est appelé théorème, il est noté comme suit: $\vdash Q$.

1.5.2 Correction/Complétude des systèmes de preuve

Définition 18 (Complétude d'un système de preuve) *Le système de preuve est dit complet: si toute formule valide A (i.e.,* $\models A$*) est démontrable (dérivable) dans le système de preuve en question (i.e.,* $\vdash A$*) .*

Définition 19 (Correction d'un système de preuve) *Le système de preuve est correct si toute formule dérivable A (i.e.,* $\vdash A$*), est valide, i.e,* $\models A$.

Remarques

1. La notation $H_1, ..., H_n \vdash Q$ signifie une déduction de Q à partir des hypothèses $H_1, ..., H_n$
2. La notation $\vdash Q$ signifie que Q est prouvable dans le système formel (i.e., que Q est déductible à partir des axiomes).

1.5.3 Méthode de résolution

Le système de preuve nommé "résolution" [25] possède deux règles d'inférences (résolution et factorisation), parfois fusionnées en une seule règle appelée ' 'résolution généralisée", ce système n'a pas d'axiomes. Les deux règles sont spécifiées ci-dessous.

Règles d'inférence

La méthode de résolution est caractérisée par deux règles d'inférences : la règle de résolution et la règle de factorisation :

Définition 20 (Règle de résolution) *soient $C_1 \equiv C \vee L, C_2 \equiv C' \vee \rceil l$ deux clauses, avec C t C' sont deux sous clauses éventuellement vides (notons qu'il y a deux littéraux complémentaires L et $\rceil L$ dans C_1 et C_2. La règle de résolution crée le résolvant (la conclusion de la règles) de C_1 et C_2 comme suit:*

$$\frac{C \vee L \qquad C' \vee \rceil l}{C \vee C'}$$

La clause $C \vee C'$ est appelée résolvant.

Définition 21 (Factorisation) *Si une clause contient deux fois le même littéral, alors la factorisation supprime une copie:*
$\frac{C \vee L \vee L}{C \vee L}$

Remarques

1. La notation $\gamma \vdash A$ indique l'existence d'un arbre de dérivation (employant les deux règles précédentes) reliant l'ensemble des formules γ et A avec :

   (a) La racine est la formule A.

   (b) Les feuilles sont les formules de l'ensemble γ.

   (c) Dans cette situation, nous disons que la formule A est dérivable à partir de l'ensemble γ.

2. Si $A \equiv \perp$ alors L'ensemble γ est réfutable ou contradictoire (insatisfaisable).

3. Pour montrer qu'une formule A est valide: nous prouvons que sa négation est contradictoire. Plus particulièrement, le principe de réfutation de la résolution consiste à remplacer la preuve de A par la considération de $\rceil A$, ensuite nous montrons que cela conduit à une contradiction.

Exemple 1.10.
La régle de résolution est illustrée comme suit:
$\{A \vee \rceil B \vee D, B \vee \rceil E\} \vdash A \vee D \vee \rceil E$.
nous avons supprimé dans cet exemple les littéraux complémentaires $\rceil B$ *et* B.

Principe de réfutation

Pour montrer que $\gamma \vdash A$, il suffit de montrer que $\gamma \cup \{\rceil A\} \vdash \square$.

Algorithme de résolution

Avant de présenter l'algorithme de résolution, nous notons que dans le cas où nous voulons prouver une formule à partir d'un ensemble d'hypothèses, alors nous appliquons d'abord le principe de réfutation ensuite nous déroulons l'algorithme mentionné ci-dessous.

Algorithme 1 : méthode de résolution

Input : F: une formule sous la FNC;
Output : 'satisfaisable' ou 'insatisaisable';
1 $CurrentSet \leftarrow \emptyset$;
2 $PrecedentSet \leftarrow \emptyset$;
3 **foreach** *clause* cl_i *de* F **do**
4 | $CurrentSet \leftarrow CurrentSet \cup \{cl_i\}$;
5 **end**
6 **while** $((|CurrentSet| \neq |PrecedentSet|) \wedge (\square \notin CurrentSet))$ **do**
7 | $PrecdentSet \leftarrow CurrentSet$;
8 | $CurrentSet \leftarrow CurrentSet \cup \{R | R$ est un résolvant de deux clauses dans CurrentSet $\}$;
9 **end**
10 **if** $\square \in CurrentSet$ **then**
11 | return 'insatisfaisable';
12 **else**
13 | return 'satisfaisable';
14 **end**

Remarques

1. La résolution n'est pas complete (dans le sens fort) parce-qu'on ne peut pas générer toutes les conséquences sémantiques (ou logiques) d'un ensemble de formules.
2. La résolution est **complete pour la réfutation**, càd que nous pouvons décider si un ensemble de formules est insatisfaisable ou non à travers la preuve par résolution, i.e., si δ est insatisfaisable, alors $\delta \vdash \square$.
3. La résolution est correcte, i.e., Si $\delta \vdash A$ alors $\delta \models A$

Exemple 1.11.
Montrer que la formule suivante est insatisfaisable:
$[(B \wedge A) \rightarrow \rceil C) \wedge ((D \vee C) \rightarrow A) \wedge (\rceil E \rightarrow (B \vee D)) \wedge (D \rightarrow E) \wedge (E \rightarrow B) \wedge (C \rightarrow (E \vee A))]$. Aprés normalisation (FNC) de chaque sous formule, nous obtenons:

- $H_1 : \rceil B \vee A \vee \rceil C$.
- La deuxième sous-formule $(\rceil D \wedge C) \vee A$ se décompose en :
  - $H_2 : A \vee \rceil D$
  - $H_3 : A \vee \rceil C$

- $H_4 : \rceil B \vee C$
- $H_5 : E \vee B \vee D$
- $H_6 : \rceil D \vee E.$
- $H_7 : \rceil E \vee B.$
- $H_8 : \rceil C \vee E \vee A.$

La figure 1.1 montre la dérivation de la clause vide à partir de $H = \{H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8\}$

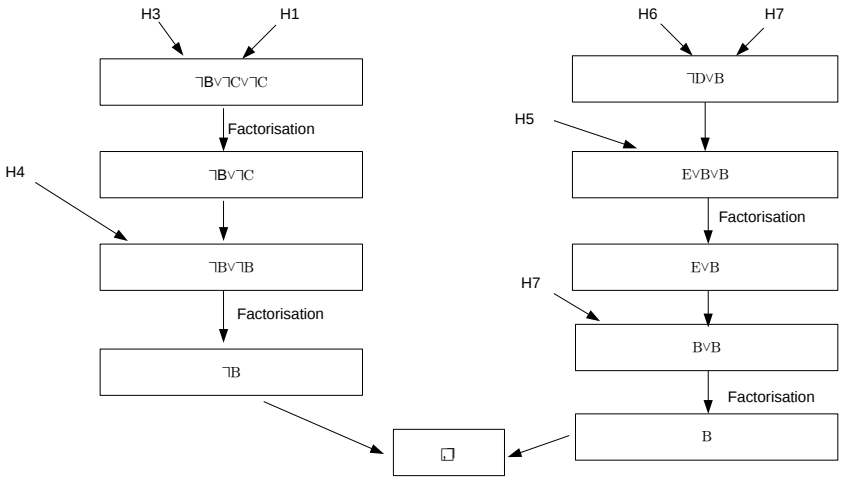

Figure 1.1: Arbre de résolution

1.6 Propriétés fondamentales

- La logique des propositions possède plusieurs systèmes de preuve qui sont **corrects et complets** (ex: système de Gentzen ou calcul de séquents [9], déduction naturelle, système d'Hilbert), et de ce fait la logique des proposition est **correcte et complete**.
- La logique des propositions est **décidable**, parce qu'il y a une procédure effective qui permet de dire en un temps fini si une formule donnée est valide (ou théorème) ou non (ex: méthode de résolution).

1.7 Problèmes SAT

Définition 22 *Le problème de Satisfaisabilité (SAT) est un problème de décision qui consiste à déterminer si un formule propositionnelle F admet, ou non, un modèle. En général, nous supposons que F est sous la forme normale conjonctive.*

La question de satisfaisabilité possède une importance majeure dans le domaine d'ingénierie; par exemple, la vérification de correction d'un programme ou circuit électronique, ou l'équivalence de deux circuits éléctroniques peut être convertie en un problème SAT. En effet, l'existance d'une erreur ou un bug dans le programme/circuit est le synonyme d'une formule satisfaisable, par contre l'absence d'erreurs veut dire que la formule est insatisfaisable. De même, nous montrons dans l'exemple 1.12, l'utilisation d'un solveur de problème de satisfaisabilité dans la verification de l'équivalence de circuits logiques. Notons aussi que les solveurs SAT peuvent résoudre efficacement des problèmes tels que la planification, la découverte de connaissances, ou les problèmes de satisfaction de contraintes [28]. Enfin, il est utile de noter que le problème de satisfaisabilité est considéré comme étant NP-Complet [5] (voir la section 3.4.2 du chapitre 3 pour plus de détail sur cette classe de complexité). La classe NP-Complet signifie que le nombre d'étapes nécessaires à la vérification d'une seule solution potentielle est polynomial (par rapport au nombre de variables), mais nous avons un nombre assez grand de solutions à vérifier (2^n), et jusqu'à maintenant, il n' y a pas d'algorithme polynomial qui fait les deux phases en même temps (i.e., vérification et recherche).

Exemple 1.12.
Supposons qu'on veut montrer que les circuits 1 et 2 montrés dans la figure 1.2 ne sont pas équivalents. La procédure à suivre consiste en 04 étapes:

1. Exprimer la sortie du circuit 1 (C1) en fonction de ses entrées (à l'aide d'une formule F1 qui sera convertie en FNC notée F'1).

2. Exprimer la sortie du circuit 2 (C2) en fonction de ses entrées (à l'aide d'une formule F2 qui sera convertie en FNC notée F'2).

3. Créer la fonction XOR dont les entrées sont les sorties des circuits C1 et C2 (i.e., X4 et X6). Le résultat du XOR est appelé par exemple X7. Remarquons que si C1 et C2 sont équivalents alors X7 vaut toujours 0, et sinon il peut valoir 1 pour certaines entrées. La relation logique entre X4 et X6 d'une part et X7 d'autre part, est exprimée sous forme d'une formule F3 qui sera convertie en FNC notée F'3.

4. Nous supposont que X7 est vrai (i.e., que C1 et C2 ne sont pas équivalents), et nous exécutons le solveur SAT sur l'expression $(F'1 \wedge F'2 \wedge\ F'3 \wedge X7)$. Si cette formule est satisfaisable, alors C1 et C2 ne sont pas équivalents, et sinon ils sont équivalents.

L'exécution de ces étapes donnera:

- La formule F1 (avant normalisation) est : $(X3 \leftrightarrow \rceil X1) \wedge ((X3 \wedge X2) \leftrightarrow X4)$.

- La formule F2 (avant normalisation) est : $((X1 \vee X2) \leftrightarrow X5) \wedge (X6 \leftrightarrow \rceil X5)$.
- La formule F3 qui représente le XOR est : $((X4 \oplus X6) \leftrightarrow X7)$.
- La formule finale (F) à vérifier sera: $(FNC(F1) \wedge FNC(F2) \wedge FNC(F3) \wedge X7)$.

Dans cet exemple, C1 et C2 ne sont pas équivalents, puisque le solveur SAT trouvera une affectation qui garantira la satisfaisabilité de F. (e.g., X1=0,X2=0,X3=1,X4=0,X5=0,X6=1,X7=1).

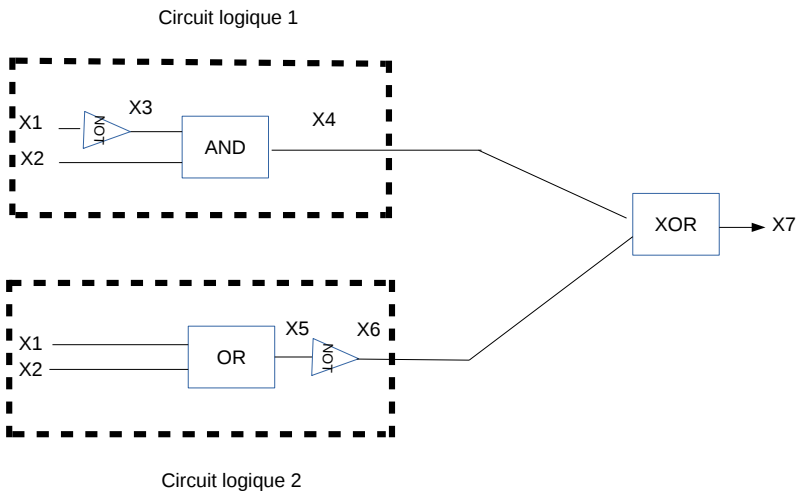

Figure 1.2: Vérification de l'équivalence des circuits logiques

1.7.1 Algorithmes complets

Les algorithmes complets permettent toujours de dire si une formule propositionnelle est satisfaisable ou non (par opposition aux algorithmes incomplets), ceci est réalisé par l'exploration integrale de l'espace de recherche de la formule d'entrée. Par conséquent, la complexité théorique de ces algorithmes est exponentielle, et ne peut garantir un délai raisonnable pour toute recherche. Dans ce qui suit, nous décrivons l'algorithme le plus fameux de cette categorie nommé aussi 'DPLL'.

Solveurs DPLL

L'algorithme DPLL (Davis-Putnam-Logemann-Loveland) [5] est creé en 1962 par les quatre chercheurs cités précédemment, il est considéré comme le noyau

de base des solveurs SAT modernes. DPLL est basé sur trois operateurs: la propagation unitaire, les littéraux purs et la division binaire. Les étapes du solveur DPLL sont expliquées ci-dessous:

Algorithme 2 : DPLL

Input : F: une formule propositionnelle sous la FNC;
Output : $Resultat \in \{False, True\}$

```
1  /*simplification de la formule*/;
2  if il existe dans F un littéral l Pur (ou monotone) then
3  |  Return DPLL (F/l);
4  end
5  /*Propagation Unitaire*/;
6  if il existe dans F une clause unitaire l then
7  |  Return DPLL (F/l);
8  end
9  /* Réfutation */;
10 if contient au moins une clause vide then
11 |  Return (False);
12 end
13 /* Satisfaisabilité */;
14 if  F ne contient aucune clause then
15 |  Return DPLL (True);
16 end
17 /* Division */;
18 v ← une variable issue de F;
19 if DPLL (F/v) = True then
20 |  Return (True);
21 else
22 |  Return ( DPLL (F/¬v));
23 end
```

Phase 1. *(Lignes 2-4)* Nous cherchons les littéraux purs (cad ceux qui ne changent pas de signes dans F), à fin de les fixer à 1, et par conséquent, on reduira le nombre de clauses ainsi que leurs tailles.

Phase 2. *(Lignes 6-8)* Nous cherchons les clauses unitaires (càd celles qui possèdent un seul littéral), à fin de les fixer à 1, et par conséquent, on reduira le nombre de clauses ainsi que leurs tailles. L'enchainement de cette règle appelé aussi 'Propagation unitaire' peut créer d'autres clauses unitaires qui peuvent être simplifiées à leurs tours avec la même règle.

Phase 3. *(Lignes 10-12)* Ce premier test d'arrêt, permet de détecter la présence d'une clause vide (une contradiction), si le résultat est positif , nous arrêtons la recherche avec un résultat 'Faux' (cad que la formule courante est insatisfaisable).

Phase 4. *(Lignes 14-16)* Ce deuxième test d'arrêt, permet de vérifier si l'ensemble de clauses est vide, si le résultat est positif, nous arrêtons la recherche avec un résultat 'Vrai' (cad que la formule est satisfaisable).

Phase 5. *(Lignes 18-23)* Cette étape permet de diviser le problème original en deux sous problèmes plus simples que le premier, elle consiste à choisir une variable notée 'v', et par la suite, nous créons deux sous problèmes notés F/v et F/¬v, dans lesquels nous remplaçons respectivement les occurrences de v dans F par 1 (resp par 0). Nous appliquons le même algorithme sur les deux sous problèmes. Notons que si l'un des deux sous problèmes retourne Vrai (cad SAT), alors nous retournons Vrai (cad SAT) pour le parent, sinon si les deux sous problèmes retournent Faux, alors nous retournons Faux (cad INSAT) pour le parent.

Exemple 1.13.
Reprenons la même formule de l'exemple 2.11 et essayons de la résoudre avec DPLL. Les clauses sont données comme suit: $H = \{H1, H2, H3, H4, H5, H6, H7, H8\}$

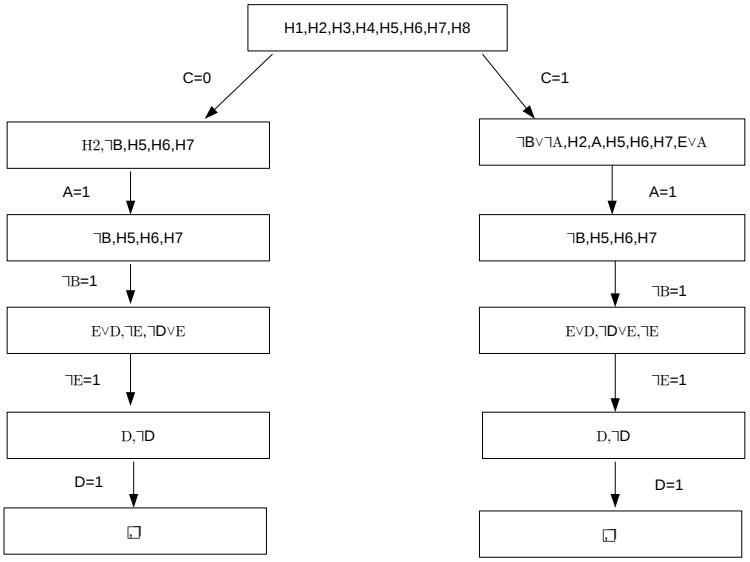

Figure 1.3: Trace de l'algorithme DPLL

Solveurs CDCL

Les solveurs sat modernes appelés aussi solveurs CDCL (conflict driven caluse learning) [1] se basent sur le sequelette DPLL tout en ajoutant la brique de l'apprentissage des clauses; plus précisement, les solveurs SAT CDCL (implémentés dans MiniSAT[7], Zchaff [22], Z3 [6] et ManySAT[14]) ajoutent les briques suivantes afin d'accélérer le temps de traitement des formules propositionnelles:

1. L'apprentissage de nouvelles clauses (no-good) à partir des conflits détectés lors de la recherche [19,32].
2. Exploitation des structures de conflicts durant l'apprentissage [19].
3. Utilisation des structures paresseuses pour la représentation des formules [22].
4. Les heuristiques de branchement doivent alléger le coût de calcul [10]; en plus, elles doivent recevoir un feedback des conflits rencontrés [22].
5. Redémarrage périodique de la recherche avec back-tracking [12].
6. Des politiques pour la suppression des clauses apprises [11,26].

Dans ce qui suit, nous donnons une version de l'algorithme CDCl inspirée de [27].

Algorithme 3 : CDCL

```
   Input : F: une formule propositionnelle sous la FNC;
   Output : Resultat ∈ {SAT, INSAT}
 1 I ← ∅, profondeur ← 0 ;
 2 while True do
 3 |   Effectuer la PU sur (Σ, I);
 4 |   if un conflit est détécté then
 5 |   |   if profondeur = 0 then
 6 |   |   |   Return INSAT ;
 7 |   |   end
 8 |   |   C ← la clause Conflit déduite;
 9 |   |   l ← l'unique littéral de C affecté à la profondeur du conflit;
10 |   |   profondeur ← Max {profondeur(x) : x ∈ C/l};
11 |   |   I ← I moins tous les littéraux assignés à une profondeur
   |   |   supérieure à profondeur;
12 |   |   (σ, I) ← (Σ∪ C, I.l);
13 |   else
14 |   |   if I est total then
15 |   |   |   Retourner SAT ;
16 |   |   end
17 |   |   Choisir un littéral l apparaissant dans Σ|I; I ← I.l; profondeur ←
   |   |   profondeur + 1
18 |   end
19 end
```

Les algorithmes CDCL (voir l'algorithme 3) acceptent deux arguments: l'ensemble des clauses σ et la solution notée I (et qui est initialement vide). Ils peuvent être vus comme une boucle infinie qui exécute les étapes suivantes [32]:

1. Enchainer la propagation unitaire afin de générer un conflit (ligne 3). Si la clause vide est dérivée à partir de la racine alors on retourne INSAT (lignes 4-7), sinon on exécute le reste des instructions.

2. Construire la clause apprise notée C (appelée aussi no-good ou clause de conflit) par résolution et en utilisant le graphe des conflits (ligne 8); Déterminer le niveau des variables de C (appelé niveau de retour) qui est immediatement plus élevé que le niveau du conflit (i.e., le niveau de l) (ligne 9-10); annulation de toutes les affectations faites aprés le niveau du retour (retour non-chronologique) (ligne 11), et ajout de la clause apprise (ligne 12). Si la clause apprise est de type FUIP (first unique implication point)[32], alors on met à jour I en affectant l à 1 (ligne 12).
3. Retourner un succés (avec le modèle) si la clause vide n'est pas générée et si l'affectation est totale (lignes 14-15).
4. Affecter un nouveau littéral de σ (l'équivalent de la division binaire de DPLL) et mettre à jour I et la profondeur de l'arbre (ligne 17) et retourner à (3).

Apprentissage de clauses et retour arrière non chronologique (clause learning)

L'idée derrière l'apprentissage des clauses est de trouver les affectations de variables qui mènent toujours à une contradiction, une fois cette affectation partielle de variables est identifisée, nous ajoutons une clause qui l'interdira (et par conséquent, ce conflit ne va plus apparaitre dans les autres régions de l'espace de recherche). La clause de conflit (ou clause apprise) est établie comme suit:

- Tout d'abord, nous construisons le graphe d'implication (voir la figure 1.4) Le graphe d'implication est un DAG (Graphe Acyclique Dirigé) crée à chaque conflit (le conflit est materialisé par une affectation antagoniste de la même variable, telle que $X12$ et $\neg X12$ dans la figure 1.4). Chaque nœud de ce DAG correspond à une variable assignée. Les arcs incidents à un noeud X signifient qu'il y a une clause $A_i \vee X$, sachant que les A_i sont les littéraux (antécédants)de X, qui sont déjà affectés à faux (par propagation unitaire ou division), et ceci force l'affectation du noeud X à vrai. Une variable de division est représentée par un nœud sans arc incident. A chaque noeud (ou littéral) du DAG est associé un numero qui représente sa profondeur (ou son niveau) d'affectation, par exemple les variables X2, X7 et X6 ont la profondeur 2, de même X4, ¬X9, X10,X11, X12 et ¬X12 ont la profondeur 4.
- Pour construire la clause de conflit, nous divisons les noeuds du DAG en deux groupes disjoints: le groupe nommé "raison" (noté GR), contenant entre autres, les variables de division, et le groupe "conflictuel" (noté GC), contenant entre autres les deux littéraux antagonistes. Différents decoupages correspondent à différentes politiques d'apprentissage. On appelle coupe ou coupure l'ensemble des arcs sortants de GR incidant à un nœud de GC. La clause apprise, appelée aussi "clause de conflit", est la disjonction de la négation des littéraux représentant les nœuds de GR ayant un de leurs arcs sortants dans l'ensemble de coupure. Dans ce qui suit, nous citons quelques exemples:
  - Si on considére GC=$\{X12, \neg, X12\}$ et GR le reste des noeuds, dans ce cas, cette coupure notée coupe1 engendre la clause apprise $\neg X5 \vee \neg X10 \vee \neg X11$.

- Si on considére GC=$\{X10, X11, X12, \neg X12\}$ et GR le reste des noeuds, dans ce cas, cette coupure notée coupe2 engendre la clause apprise $\neg X5 \vee X9$.
- Si on considére GC=$\{\neg X9, X10, X11, X12, \neg X12\}$ et GR le reste des noeuds, dans ce cas, cette coupure notée coupe3 engendre la clause apprise $\neg X5 \vee \neg X4 \vee \neg X6 \vee \neg X7$.
- Si on considére GR=$\{X1, X2, X4\}$ (notons que X3 n'est pas impliqué dans le conflit et par conséquent ne fait pas partie de GR) et GC le reste des noeuds, dans ce cas, cette coupure notée coupe4 engendre la clause apprise $\neg X1 \vee \neg X2 \vee \neg X4$, dans cet exemple toutes les variables de division qui ont un lien (direct ou indirect) avec le conflit font partie de GR (en général, ce schéma d'apprentissage est moins efficace parce qu'il y a qu'un sous enemble de variables de division qui sont réelement pertinentes pour l'apprentissage).

Dans le solveur relast, les chercheurs placent dans l'ensemble GR tous les nœuds affectés à une profondeur inférieure à la profondeur courante, en plus de la variable de division. Sur l'exemple précédent, relsat apprendera la clause $(\neg X6 \vee \neg X7 \vee \neg X4)$.

Dans la pratique, il est préférable que la clause apprise soit de courte taille (à fin de simplifier la propagation unitaire). Dans cette optique, les chercheurs ont défini le concept point d'implication unique (UIP), un UIP est un nœud de la profondeur de décision courante (cad il est affecté à la suite à la dernière décision), en plus, tout chemin entre la variable de décision courante et les deux variables opposées (dans notre exemple $X11, \neg X11$) doit passer par le noeud UIP. (Notons que la variable de décision courante est un UIP valable). Le UIP le plus proche des deux variables du conflit est appelé first unique implication point (FUIP). En considérant l'ensemble de tous les noeuds affactés à la profondeur courante et après le FUIP comme GC, on construira une clause de conflit correspondante que l'on notera ClauseFUIP (dans l'exemple courant GC=$\{X11, X12, \neg X12\}$, en plus: la clause ClauseFUIP - associée à la coupure coupe5 - vaut $(\neg X5 \vee \neg X10)$. Notons que ClauseFUIP est utilisée dans plusieurs solveurs SAT modernes tels que GRASP [19,20], Chaff [22] et minisat [7]. Il a été montré expérimentalement que ce schéma d'apprentissage (basé sur ClauseFUIP) est le plus efficace sur les problèmes industriels.

Heuristiques des choix de variables de décision

La littérature nous offre une multitude de méthodes qui permettent de sélectionner les bonnes variables de décision. En général, elles favorisent les variables qui ont plus d'occurences dans les clauses et surtout lorsque les clauses ont une courte taille (e.g., l'heuristique "Maximum Occurrences in clauses of Minimum Size"(MOMS) [5]). Parfois, il sera judicieux de donner des poids différents aux clauses qui ont des tailles différentes (les poids les plus grands seront attribués aux clauses de taille une ou deux). En plus, il y a des heuristiques telles que "Variable State Independent Decaying Sum" (VSIDS) [22] qui favorisent les variables qui se présentent dans les clauses de conflits et surtout les conflits récents.

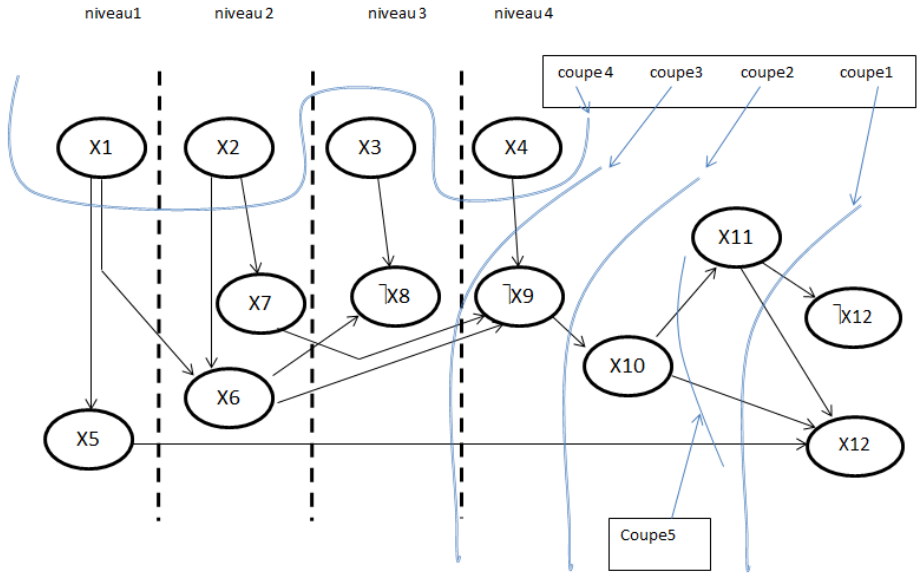

Figure 1.4: Graphe de conflit

Littéraux observés (watched litterals)

Nous constatons que la propagation unitaire occupe une large partie du temps d'exécution des solveurs existants (et elle peut aller jusqu' à 90 %) [27], donc si nous pouvons réduire le temps de recherche des clauses unitaires (celles qui ne possèdent que des littéraux faux sauf un), nous améliorons largement les performances des solveurs SAT: l'objectif des littéraux observés (watched literals)[22] est d'eviter l'inspection des clauses:

1. qui ont un littéral satisfait (affecté à 1), puisque ces clauses sont déjà satisfaites.

2. qui ont deux littéraux non affectés, parce qu'elles ne peuvent être unitaires.

L'astuce des littéraux observés consiste à choisir deux littéraux (distincts) non affectés de chaque clause, une clause doit être examinée lorsque un littéral observé devient faux. L'heuristique est resumée comme suit:

- Après une affectation d'une variable dans l'arbre de recherche, si cette variable n'est pas observée, on ne fera pas la propagation unitaire (on gagnera le temps de recherche des clauses unitaires et même le temps de défaire les affectations en cas de retour arrière).

- Si la variable affectée est observée dans une clauce C_i alors:

  - Si le littéral correspondant à cette variable devient vrai dans C_i, alors C_i est satisfaite.
  - Si le littéral correspondant à cette variable devient faux dans C_i, alors on cherchera un autre littéral (non affecté ou valant 1) dans C_i et il

deviendra le nouveau littéral observé et on continue l'exploration de l'arbre de recherche. Si ce littéral n'existe pas et le deuxième littéral observé est non affécté, alors C_i est unitaire et on affectera à 1 le deuxième littéral observé et on lance la propagation unitaire. Si ce littéral n'existe pas et le deuxième littéral observé est vrai, alors C_i est satisfaite.

Exemple 1.14.

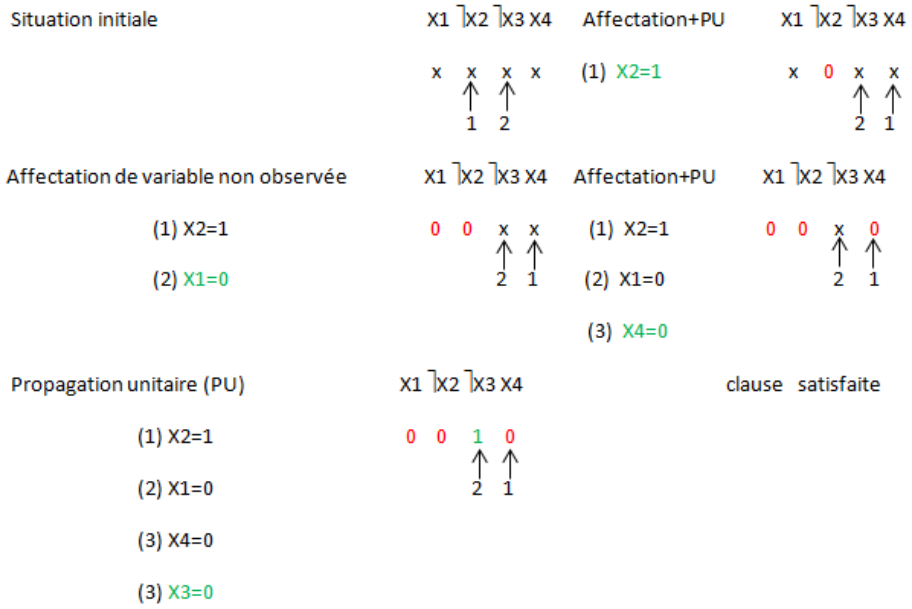

Figure 1.5: Littéraux observés

Dans la première ligne de l'exemple 1.13, nous observons les littéraux $\neg X2, \neg X3$ de la clause $(X1 \vee \neg X2 \vee \neg X3 \vee X4)$, et puisqu'ils sont pas affectés, on ne peut faire la propagation unitaire (PU) sur cette clause. Ensuite l'algorithme CDCL choisira X2 comme variable de division (X2=1), et par conséquent on doit choisir une deuxième variable observée (dans ce cas, on a opté vers X4) et de même, on ne peut faire la PU (puisque les variables observées ne sont pas affectées).
Dans la deuxième ligne, le solveur CDCL a choisi X1 comme variable de division (X1=0) et de même, on ne peut faire la PU, ensuite le solveur a choisi X4 comme variable de division (X4=0), et donc la clause est devenue unitaire parce qu'il y a qu'une seule variable observée non affectée, et par conséquent on applique la PU dans la troisième ligne, et ceci force l'affectationde X3 à 0 et la clause est satisfaite.

Remarque

On assiste actuellement à plusieurs extensions de solveurs SAT et prouveurs de Théorèmes qui permettent d'incorporer des théories non propositionnelles (e.g., arithmetique, egalité, vecteurs de bits, et autres), nous citons à titre d'exemple

le solveur Z3 [6] de Microsft et le prouveur de théorèmes Coq[1] d'INRIA.

1.7.2 Algorithmes incomplets

Il existe une catégorie d'approches (par exemple, walksat[21], le recuit simulé[16], ou les algorithmes génétiques adaptés aux problèmes SAT [8,18,21]), qui sont très efficaces mais incomplets. Ces derniers ne parcourent pas l'ensemble des solutions et peuvent donc ne pas répondre. Ils ont l'avantage de localiser rapidement un modèle lorsqu'il existe, mais il n'en trouvent pas toujours (même si la formule est satisfiable). En contre partie, si la formule est INSAT, nous n'aurons aucune réponse. En général les algorithmes incomplets se base sur la recherche locale stochastique (rls) [15] pour explorer l'espace de recherche. La rls permet de développer des solutions (affectations) en se basant sur une relation de voisinage et des choix guidés par l'aléatoire, les solutions acceptées dans le futur sont celles qui optimisent une fonction coût prédéfine.

Algorithme du WalkSat (optimisation stochastique): initialement, nous choisissons une interprétation aléatoire et tant que le modèle n'est pas trouvé, nous sélectionnons une clause violée et nous inversons l'une de ses variables (avec une probabilité p), sinon nous inversons une variable de la même clause qui permet de minimiser le nombre de clauses violées (pour éviter les minima locaux). Ce processus est répeté jusqu'à l'épuisement de toutes les itérations.

Le récuit simulé (RS): cet algorithme est de type recherche locale, en démarrant avec une interprétation aléatoire, RS tire une solution voisine de l'affectation courante et l'accepte si elle est meilleure que la première (en termes d'une fonction objectif à minimiser que l'on appelle énegie). Dans le cas contraire, elle est aussi acceptée avec une probabilité p qui est inversement corrélée avec la différence d'énergie (notons aussi l'existance d'un paramétre appelé température [2] et qui permet de controler p).

Algorithmes génétiques: Ces algorithmes évolutionnaires sont basés sur la sélection naturelle. Initialement, la population des interprétations est generée aléatoirement, ensuite la boucle principale de l'algorithme met à jour la population par emploi de croisement, mutation et re-sélection. Ce processus permet de garder les solutions les plus intéréssantes. Le lecteur intéréssé trouvera plus d'informations sur les algorithmes évolutionnaires dans la reférence [13].

1.8 Conclusion

Nous avons étudié dans ce chapitre le langage de logique proposistionnelle, ce dernier possède des propriétés attirantes en termes de décidabilité et complétude; en contre partie, ce langage possède un faible pouvoir d'expression, par exemple, il est impossibe d'exprimer des phrases contenant des quantificateurs (tels que certains ou tous), et de ce fait, l'applicabilité des algorithmes de démonstration de cette logique reste toujours limitée dans certains problèmes du monde réel.

[1] The Coq Proof Assistant.http://coq.inria.fr/.

[2] La température est une variable qui controle la diversification de la recherche en autorisant l'évitement des minima locaux, elle est baissée à chaque itération, plus la température est grande plus p est grand, et plus la temperature est petite plus p est faible.

1.9 Exercices

1.9.1 Exercice 1

Vérifier si les conséquences sémantiques (logiques) suivantes sont établies ou non:

- $\models (A \leftrightarrow B) \leftrightarrow ((B \leftrightarrow C) \leftrightarrow (A \leftrightarrow C))$.
- $\{D \rightarrow B, D \rightarrow F, \rceil(B \wedge F), (E \vee D)\} \models E$.

1.9.2 Exercice 2

Montrer par résolution que :

- $[(A \wedge C) \rightarrow F) \wedge ((D \wedge \rceil E) \rightarrow F) \wedge (A \rightarrow (C \vee D)) \wedge (F \rightarrow E) \wedge (A \rightarrow \rceil E)] \vdash \rceil A$.
- $[((B \wedge F) \rightarrow (A \vee E)) \wedge (\rceil C \rightarrow (E \vee F)) \wedge ((B \wedge A) \rightarrow C) \wedge (E \rightarrow \rceil B) \wedge (C \rightarrow D) \wedge B] \vdash D$.

1.9.3 Exercice 3

Nous voulons colorier la carte géographique montrée dans la figure 1.6 à l'aide de 03 couleurs (rouge, vert, bleu), sachant que deux Wilaya adjacentes ne doivent pas avoir la même couleur. On vous demande de représenter ce problème à l'aide de logique propositionnelle. (Il faut associer un atome à chaque paire (wilaya, couleur), ensuite il faut coder les contraintes sous forme de formules ou clauses).

Figure 1.6: Carte géographique

1.9.4 Exercice 4

Modéliser la contrainte lineaire suivante avec la logique des propositions (les variables X_i sont binaires).

- $X_1 + X_2 + X_3 + X_4 = 1$.
- $X_1 + X_2 + X_3 + X_4 < 3$.

1.9.5 Exercice 5 (sans corrigé)

Modéliser le problème du Sudoku avec la logique propositionnelle, sachant que la grille contient n lignes, n colonnes, $\sqrt{n}$ blocs.

1.9.6 Exercice 6 (sans corrigé)

Montrer par résolution et DPLL que la formule suivante est insatisfaisable: $[((B \wedge F) \rightarrow (A \vee E)) \wedge (\neg C \rightarrow (E \vee F)) \wedge ((B \wedge A) \rightarrow C) \wedge (E \rightarrow \neg B) \wedge (C \rightarrow D) \wedge B \wedge \neg D]$

1.9.7 Exercice7 (QCM)

Choisir la réponse correcte:

- L'ensemble des formules satisfaisables englobe celui des formules contingentes [V/F]?
- Les instances (formules) 2SAT sont plus faciles que les instances 3SAT[V/F]?
- Toutes les instances (formules) 3SAT sont difficiles (en termes de temps de processing)[V/F]?
- pour résoudre le problème de variabilité du temps d'exécution d'une formule 3SAT, on effectue:
  1. **(a)** Une exécution intensive des littéraux purs.
  2. **(b)** Un redémarrage aléatoire.
  3. **(c)** Une surveillance des littéraux observés.
  4. **(d)** Une recherche des variables critiques (ou Back-doors).
- L'apprentissage des clauses permet de ne pas refaire les mêmes erreurs (mauvaises affectations de variables) dans d'autres régions de l'espace de recherche[V/F]?
- La méthode de conséquence sémantique est en moyenne plus efficace (rapide) que la méthode de résolution [V/F]?
- les littéraux observés permettent:
  1. **(a)** L'accélération de l'exécution des littéraux purs.
  2. **(b)** L'accélération de l'exécution des clauses unitaire.
  3. **(c)** L'accélération de la division binaire.

4. **(d)** L'accélération de la recherche des variables critiques.

- Le retour non chronologique permet de trouver et modifier les causes racines du conflit actuel[V/F]?

Chapitre 2

Logique des prédicats

2.1 Introduction

Dans la logique d'ordre 0 (premier chapitre), les propositions simples ne sont pas divisibles; elles sont soit vraies ou fausses. Par conséquent, si un seul composant de la proposition change, nous sommes obligés de changer le nom de la nouvelle proposition et nous ne pouvons plus réutiliser les autres ingrédients. Dans la logique d'ordre 1, nous introduisons le concept de prédicat qui permet de modéliser les énoncés ayant la forme sujet-prédicat ou sujet-prédicat-complément (voir les exemples montrés ci-dessous). Un prédicat est une relation liant un ou plusieurs arguments, et sa valeur de vérité dépend des instances prises par ses arguments.

Exemple 2.1.
La pharse suivante est modélisée avec deux variables propositionnelles différentes malgré les éléments partagés:
"Ali aime les poissons et n'aime pas les poulets". Cette expression est traduite en logique propositionnelle comme suit: P1 $\wedge$ P2 (il est évident que P2 partage le sujet et le verbe avec P1)

Par opposition à la logique proposistionnelle, la logique des prédicats (ou d'ordre 1, notée ausi LPRED), décompose les propositions en :

- Un sujet et éventuellement des compléments.
- Un prédicat [1] (la propriété du sujet ou le verbe de la phrase)

Exemple 2.2. "Ali est intelligent" notée formellement **intelligent(Ali)** est decoupée en:

- Un sujet : "Ali"
- Un prédicat : "est intelligent"

La proposition "Ali corrige l'exercice" notée formellement **corrige(Ali, exercice)** est decoupée en:

[1] Un prédicat est une relation liant un ou plusieurs objets. Si le nombre d'arguments du prédicat est n, alors le prédicat est un sous ensemble du produit cartesien $D_1 \times D_2 \times ..., D_n$, avec $D_1, ..., D_n$ sont les domaines de définition des arguments.

- Un sujet "ali" et un complément "l'exercice"
- Le prédicat : "corrige"

En plus, la logique des prédicats permet l'expression des quantifications [2] (universelles ou existentielles):

Exemple 2.3.

- "Tous les etudiants sont riches" notée formellement $\forall x(etudiant(x) \rightarrow riche(x))$
- "certain etudiants sont riches" notée formellement $\exists x(etudiant(x) \wedge riche(x))$

La notion de fonction constitue une autre nouveauté de la logique des prédicats, elle permet d'associer à un ou plusieurs individus (représentant les arguments) un individu unique (qui représente le résultat). Par exemple, la fonction pere(x) permet d'associer des résultats uniques à des individus en entrée. (e.g., pere(reda)=yacine).

2.2 Synatxe

2.2.1 Grammaire et vocabulaire

Définition 23 (Vocabulaire) *L'alphabet (vocabulaire) de la logique des prédicats est défini comme suit:*

1. *l'ensemble des variables denotées x , y , z , $\cdots$*
2. *l'ensemble des constantes denotées a , b , c , $\cdots$*
3. *l'ensemble des fonctions denotées f , g , h , $\cdots$*
4. *l'ensemble des prédicats denotés p , q , r , $\cdots$*

Le nombre d'arguments d'une fonction ou d'un prédicat est appelé arité. Une fonction avec une arité nulle est considérée comme une constante.

Définition 24 (Grammaire) *L'ensemble des formules bien formées (fbf) de la logique des prédicats (LPRED) est défini comme suit:*

- *Une fbf est soit:*
  - *Un atome (l'atome est un prédicat qui possède des termes comme arguments, la défintion des termes est donnée par la suite).*
  - *Une expression de la forme*
    - $\forall$ *variable fbf*
    - $\exists$ *variable fbf*
    - $\neg$ *fbf*
    - *fbf connecteur fbf*
-

[2] La quantification permet de dire qu'une partie d'un ensemble (ou tous les élements de l'ensemble) vérifie (ent) une certaine propriété (ou prédicat).

- *Un terme est soit:*
  - *Une constante*
  - *Une variable*
  - *Une fonction qui a d'autres termes comme arguments*
- *Un connecteur est un:*
  - $\wedge$
  - $\vee$
  - $\rightarrow$
  - $\leftrightarrow$

2.2.2 Priorité des connecteurs

la priorité des connecteurs permet de désambiguiser les étapes du calcul de degré de vérité d'une formule, elle est donnée comme suit:

- $\forall$, $\exists$(la plus haute priorité)
- $\neg$
- $\wedge$,$\vee$
- $\rightarrow$, $\leftrightarrow$ (la plus basse priorité)

Définition 25 (Variables libres/liées) *Soit F une fbf de la logique des prédicats et x une variable possèdant plusieurs occurrences dans F. Une occurrence de x est dite liée dans F, si elle est dans la portée d'un quantificateur* [3] $\forall$ *ou* $\exists$*. Les occurences de variables qui apparaissent dans une formule sans quantification sont dites libres. Une variable dont toutes les occurrences sont liées est dite liée, sinon elle est dite libre.*

Exemple 2.4.

- Dans la formule $\exists x p(y,x) \vee q(x,y)$, la première occurrence de x est liée alors que la deuxième est libre, les deux occurrences de y sont libres.
- Dans la formule $\forall x(r(x,y) \rightarrow \forall y q(x,y))$, les deux occurrences de x sont liées, alors que la première occurrence de y est libre et la deuxième est liée.

Remarques
Toutes les variables d'une fbf sans quantificateur sont libres, en plus, si x est libre dans A, x est liée dans $\forall xA$ et dans $\exists xA$.

Définition 26 (substitution) *Soit F une fbf contenant une variable x et soit t un terme. Une substitution de x dans F est le résultat du remplacement simultané de toutes les occurrences libres de x dans F par le terme t. La substitution est notée* $\{x/t\}$*, elle peut se généraliser pour plusieurs paires variables termes (e.g.,* $\{x_1/t_1, \cdots, x_i/t_i\}$*)*

[3] si F $\equiv (\forall xA)$ ou F $\equiv (\exists xA)$, alors la formule A est appellé la portée du quantificateur $\forall$ ou $\exists$.

Exemple 2.6.

- La substitution de x par g(z) dans la formule $\exists x\ p(z,x) \vee q(x,y)$, donnera $\exists x\ p(z,x) \vee q(g(z),y)$.
- La substitution de y par g(z) dans la formule $\forall x(r(x,y) \rightarrow \forall y\ q(x,y))$, donnera $\forall x(r(x,g(z)) \rightarrow \forall y\ q(x,y))$.

Définition 27 (Formule close ou fermée) *Une formule est dite close si elle ne contient pas de variables libres, sinon elle est dite ouverte.*

2.3 Théorie des modèles (Sémantique)

L'interprétation d'une fbf $\in$ LPRED consiste à calculer sa valeur de vérité (donc le sens d'une fbf est tout simplement sa valeur de vérité). Pour mener cette tâche, nous devons tout d'abord interpréter (donner un sens) aux ingrédients de la formule logique et qui sont:

- Les variables libres.
- Les constantes.
- Les atomes (ou prédicats).
- Les fonctions.

Après l'interprétation des ingrédients de base, nous utilisons des règles pour interpréter les fbf complexes (et qui mettent en jeu des constructeurs tels que $\forall, \exists, \neg, \rightarrow \cdots$)

Définition 28 (Interprétation) *Une interprétation I est définie par le couple suivant:*

- *Un ensemble non vide D appelé domaine d'interprétation.*
- *Une fonction d'interprétation IF qui associe*
  - *à chaque variable libre une valeur de D.*
  - *à chaque constante une valeur de D.*
  - *à chaque symbole de fonction à n arguments une fonction de D^n dans D.*
  - *à chaque symbole de prédicat à m arguments une relation $m-aire$ (i.e., un sous-ensemble de D^m).*

Exemple 2.7.
Soit la formule $F \equiv p(a,x) \vee q(x,g(a))$, et soit l'interprétation I =(D,IF) définie par:

- D=$\{1,2\}$
- IF est spécifié comme suit:
  - IF(a)=2.

- IF(x)=1.
- IF(g)=$\{(1,1),(2,1)\}$.
- IF(p)=$\{(1,1),(1,2)(2,1),(2,2)\}$.
- IF(q)= $\emptyset$.

En appliquant I sur les ingrédients de F, nous aurons:

- IF(g(a))=1, puisque l'image de 2 par g est égale à 1
- IF(p(a,x)= 1, puisque (IF(a),IF(x))$\in$ IF(p)
- IF(q(x,g(a)))=0, puisque (IF(x),IF(g(a)))$\notin$ IF(q)
- et enfin: IF(F)=1 puisque IF(p(a,x)= 1.

Exemple 2.8.
Reprenons la même interprétation définie dans l'exemple précédent et appliquons la sur les formules $F1 \equiv \forall x(p(g(x),a) \vee q(g(x),g(a))$, et $F2 \equiv \exists x(q(g(x),g(a)) \rightarrow p(g(x),a)$.

- F1 sera vraie si toutes valeurs possibles de x (i.e., 1 et 2) vérifient la disjonction, en particulier nous vérifions cela comme suit:
  - si x=1: IF(p(g(x),a))=1 puisque (IF(g(x)),IF(a))$\in$ IF(p) et par conséquent IF(F1)=1.
  - si x=2: IF(p(g(x),a))=1 puisque (IF(g(x)),IF(a))$\in$ IF(p) et par conséquent IF(F1)=1.
- F2 sera vraie si au moins une valeur de x (i.e., 1 ou 2) vérifie l'implication (et elle est fausse si les deux violent l'implication). En particulier, si x= 1 alors IF(q(g(x),g(a)))=0, puisque IF(q) est vide, et IF((p(g(x),a)=1 (déjà expliqué ci-dessus). Donc l'implication est vraie et IF(F2)=1.(inutile de vérifier le cas x=2, puisque F2 est quantifiée existentiellement).

Définition 29 (Modèle/Conséquence sémantique) *Une fois la notion d'interprétation est établie, nous pouvons définir les concepts de modèle et de conséquence sémantique:*

- *Soit F une formule de LPRED, un modèle de F est une interprétation I qui assure la valeur de vérité 1 à F.*
- *A est une conséquence sémantique de B ssi: tout modèle de B est aussi un Modèle de A, on note :B $\models$ A.*

Exemple 2.9.
Montrer que la formule F2: $\forall x \forall y(p(x,y) \wedge q(x,y))$ n'est pas une conséquence sémantique de F1: $\forall x \forall y(p(x,y) \vee q(x,y))$. En considérant l'interprétation I=(D=$\{1,2\}$,IF), avec IF(P)=$\{(2,1),(1,1)\}$, IF(Q)=$\{(1,2),(2,1),(2,2)\}$, nous concluons que F1 est vraie dans I (i.e., I$\models$ F1) et F2 est fausse dans I, ainsi la relation de conséquence sémantique entre F1 et F2 n'est pas établie.

Remarque

Table 2.1: Table de vérité de F1 et F2

y	x	IF(p(x,y)	IF(q(x,y)	IF(p(x,y)∨ q(x,y))	IF((p(x,y)∧ q(x,y))
1	1	1	0	1	0
2	1	1	1	1	1
1	2	0	1	1	0
2	2	0	1	1	0

- Il est impossible de vérifier la conséquence sémantique en LPRED puisque le nombre d'interprétations est infini , Par contre il est possible de prouver que cette relation est violée avec un contre exemple.
- Vu que la conséquence sémantique n'est pas pratique pour des montrer les relations logiques entre formules, nous sommes obligés de faire recours aux méthodes de preuve pour établir les relation de déductibilité entre formules. Dans ce polycopié, nous présentons la méthode de résolution qui constitue la méthode la plus efficace en LPRED.

2.4 Formes normales

Le but des formes normes normales est de transformer les formules de LPRED en un format syntaxique qui est acceptable par les méthodes de preuve. En l'occrrence, nous citons la méthode de résolution qui exige que les formules d'entrée soient sous la forme clausale.

2.4.1 Forme prénexe

Définition 30 (Forme normale prénexe) *Une formule de LPRED est sous la forme normale prénexe si et seulement si elle s'écrit:*
$\square x_1..\square x_n F$
Avec $\square$ *est un* $\exists$ *ou* $\forall$ *et* F *est une formule sans quantificateur.*

Algorithme de mise en forme prénexe

1. Éliminer les connecteurs $\rightarrow$ et $\leftrightarrow$.
2. Transporter les négations devant les atomes en utilisant ($\neg\neg F \leftrightarrow F$) et les lois de Morgan.
3. Transporter les quantificateurs en tête de la formule. (Il faut renommer les variables quantifiées plus d'une fois, pour pouvoir utiliser les règles de l'étape 3).

Règles de manipulation des quantificateurs

- $\neg(\exists x F) \leftrightarrow \forall x \neg F$
- $\neg(\forall x F) \leftrightarrow \exists x \neg F$
- $\forall x \forall y F \leftrightarrow \forall y \forall x F$
- $\exists x \exists y F \leftrightarrow \exists y \exists x F$

- $\forall x F \wedge \forall x H \leftrightarrow \forall x(F \wedge H)$
- $\exists x F \vee \exists x H \leftrightarrow \exists x(F \vee H)$
- si H ne contient aucune occurrence libre de x alors:
  - $(\forall x F) \wedge H \leftrightarrow \forall x(F \wedge H)$
  - $(\exists x F) \wedge H \leftrightarrow \exists x(F \wedge H)$
  - $(\forall x F) \vee H \leftrightarrow \forall x(F \vee H)$
  - $(\exists x F) \vee H \leftrightarrow \exists x(F \vee H)$

Exemple 2.10.
Considérons la formule F $\equiv \forall x \exists y\ p(x, z, g(y)) \rightarrow \exists z\ q(z, x)$.

- En premier lieu, nous renommons les variables liées qui présentent des conflits avec d'autres variables libres ou liées: $\forall x' \exists y\ p(x', z, g(y)) \rightarrow \exists z'\ q(z', x)$.
- En deuxième lieu, nous eliminons l'implication: $\exists x' \forall y \neg p(x', z, g(y)) \vee \exists z' q(z', x)$.
- En dernier lieu, nous décalons les quantificateurs à gauche puisqu'il n'y a pas de variables libres qui risquent de changer leur statut: $\exists x' \forall y \exists z' (\neg p(x', z, g(y)) \vee q(z', x))$.

Remarque
Toute formule de LPRED est équivalente à une formule sous la forme normale prénexe.

2.4.2 Forme de Skolem

Définition 31 (Forme de Skolem) *Une formule F est sous la forme de Skolem ssi:*

- *F est sous la forme prénexe*
- *Tous les quantificateurs existentiels sont éliminés et traités selon les règles suivantes:*
  - *R1: Si F est de la forme* $\forall X1 \forall X2 ... \forall Xn \exists Y\, A(x1, X2, ...Xn, Y)$*, elle devient:* $\forall X1 \forall X2 ... \forall Xn A(X1, X2, ...Xn, g(X1, X2, ...Xn))$
  - *R2: Si F est de la forme* $\exists Y \forall X1 \forall X2 ... \forall Xn\ A(X1, X2, ...Xn, Y)$*, elle devient:* $\forall X1 \forall X2 ... \forall Xn\ A(X1, X2, ...Xn, Y)$*, sachant que a représente une constante.*

Algorithme de skolimisation

1. Mise en forme normale prénexe.
2. Elimination des quantificateurs $\exists$ grâce aux règles R1 et R2.

Exemple 2.11.
Considérons la formule F $\equiv \exists w \forall x \exists y(p(x, z, g(y)) \wedge q(y, w))$.

- En premier lieu, nous s'assurons que F est sous la forme prénexe.
- En deuxième lieu, nous éliminons $\exists w$: $\forall x \exists y(p(x, z, g(y)) \wedge q(y, a))$.
- En dernier lieu, nous éliminons $\exists y$: $\forall x(p(x, z, g(h(x))) \wedge q(h(x), a))$.

Remarque
Toute formule de LPRED est équisatisfaisable [4] à une formule sous la forme de Skolem.

2.4.3 Forme clausale

Dans le but d'étendre la forme normale conjonctive pour la logique des prédicats, nous définissons les concepts de base présentés ci-dessous:

Définition 32 (Littéral/Clause/FNC)

- *Un littéral est un atome ou sa négation (i.e., $r(t_1, .., t_n)$ou $\neg r(t_1, .., t_n)$).*
- *Une clause est une formule de la forme $L_1 \vee .. \vee L_j$, $q \geq 0$, où chaque L_i est un littéral. La clause vide notée $\square$ représente la valeur $\perp$.*
- *Une formule est en forme normal conjonctive (FNC) ssi elle est de la forme: $C_1 \wedge .. \wedge C_m$, $m \geq 0$, où chaque C_j représente une clause.*

La forme clausale (La FNC étendue à LPRED) élimine les variables libres pour autoriser le raisonnement, ensuite elle fait appel aux formes prénexes et Skolem, et enfin elle crée les clauses en distribuant le $\wedge$ sur le $\vee$.

Algorithme de mise en forme clausale

- Entrée : une formule F $\in LPRED$
- Sortie : une formule en forme normale clausale

1. Pour toute variable libre x de F: fermer F existentiellement : remplacer F par $\exists xF$. (Le résultat de fermeture appelé F1 est equisatisfaissable avec F).
2. Mettre F1 en forme normale de Skolem (le résultat est appelé F2).
3. Mettre la matrice de F2 en forme normale conjonctive (FNC), pour ce faire, nous distribuons le $\vee$ sur le $\wedge$. Le résultat est nommé F3 $\equiv \forall X_1 ... \forall X_m(C_1 \wedge .. \wedge C_m)$ et F3$\leftrightarrow$F2

[4] F est équisatisfaisable à H ssi F est satisfaisable alors H est aussi et vice versa

4. Renommer les variables communes des clauses C_i de F3 de telle sorte que chaque clause C_i a ses propres variables. Les nouvelles clauses sont denotées CL=$\{C'_1, ..C'_m\}$
5. Retourner CL.

Exemple 2.12.
Considérons la formule F $\equiv \forall x \exists w \forall y (p(x,y) \rightarrow (r(w) \wedge q(y,w))$.

- En premier lieu, nous calculons la forme de Skolem:$\forall x \forall y (\neg(p(x,y) \vee (r(g(x)) \wedge q(y,g(x)))$.
- En deuxième lieu, nous calculons la FNC : $\forall x \forall y [(\neg p(x,y) \vee r(g(x)) \wedge (\neg p(x,y) \vee q(y,g(x))]$
- En dernier lieu, nous renommons les variables partagées entre clauses, et nous omettons tous les quantificateurs: $[(\neg p(x,y) \vee r(g(x)) \wedge (\neg p(x1,y1) \vee q(y1,g(x1))]$
- nous retournons l'ensemble des clauses CL=$\{\neg p(x,y) \vee r(g(x)), \neg p(x1,y1) \vee q(y1,g(x1))\}$

Remarque
Toute formule de LPRED est équisatisfaisable à une formule sous la forme clausale.

2.5 Méthode de résolution

Pour étendre la règle de résolution en LPRED, nous devons introduire le concept d'unfication. En effet, pour effectuer la résolution sur deux littéraux $r(t_1, .., t_m)$ et $\neg r(u_1, .., u_m)$, non seulement les deux littéraux doivent avoir le même nom, mais aussi il faut unifier tous les arguments t_j et u_j.

Définition 33 (Unification d'atomes) *Deux atomes sont unifiables s'il existe une substitution des variables par des termes qui rend identique les deux formules atomiques, cette substitution est appelée unificateur.*

Exemple 2.13.
Les atomes $p(x,a)$ et $p(f(z),y)$ sont unifiables puisqu'ils ont le même nom, le même nombre d'arguments et les arguments sont unifiables deux à deux:

- $x = f(z)$ est unifiable avec succés, puisqu'il suffit de remplacer x par $f(z)$ (i.e., $< x/f(z) >$).
- $a = y$ est unifiable avec succés, puisqu'il suffit de remplacer y par a (i.e.,$< y/a >$).
- En conclusion, la substitution $\sigma =< x/f(z), y/a >$ est l'unificateur des deux atomes.

D'autre part, nous remarquons que les atomes q(y,a) et q(f(z),b) ne sont pas unifiables, puisque l'équation a=b n'est pas unifiable (on ne peut rempalcer une constante par une autre).

De la même manière, nous définissons les règles d'unification des termes et des atomes dans l'algorithme montré en section 2.5.1.

2.5.1 Unification

Algorithme d'unification

- entrée : un ensemble fini E d'équations entre termes
- sortie : échec, ou bien une substitution notée σ
- début:
  - $\sigma =<>$.// la substitution est initialement vide.
  - Tant qu'il y a une équation de E non résolue:
    - Choisir une équation de E.
    - Appliquer une des règles suivantes à cette équation :
      1. Si elle est de la forme t = t, alors on la supprime.
      2. Si elle est de la forme $f(t_1, ..., t_n) = g(t'_1, .., t'_m)$ et f et g sont différentes, alors échec.
      3. Si elle est de la forme $f(t_1, ..., t_n) = f(t'_1, .., t'_n)$, alors on la remplace par les n équations $t1 = t'_1, .., t_n = t'n$.
      4. Si elle est de la forme t = x et t n'est pas une variable, alors on la remplace par x = t.
      5. Si elle est de la forme x = t et x apparaît dans t, alors échec.
      6. Si elle est de la forme x = t et x n'apparaît pas dans t, alors remplacer x par t dans toutes les équations de E. Mettre à jour σ avec la paire x/t.
  - rendre la substitution σ.
- fin

Remarque
L'algorithme d'unification se termine toujours, en donnant soit une substitution, soit un échec.

2.5.2 Règles d'inférence

Soient C et C' deux clauses qui n'ont pas de variables communes. La règle de résolution peut être spécifiée comme suit:
$\frac{C \vee r(t_1,..,t_n) \qquad C' \vee \rceil r(t'_1,..,t'_n)}{\sigma(C \vee C')}$
Avec, σ est l'unificateur de $r(t_1, .., t_n)$ et $r(t'_1, .., t'_n)$. La clause $\sigma(C \vee C')$ est appelée résolvant, elle crée par application de la substitution σ sur la clause $C \vee C'$.
Nous définissons la règle de factorisation comme suit:
$\frac{C \vee r(t_1,..,t_n) \vee r(t'_1,..,t'_n)}{\sigma'(C \vee r(t_1,..,t_n))}$
Avec, σ' est l'unificateur de $r(t_1, .., t_n)$ et $r(t'_1, .., t'_n)$.
Exemple 2.14.

La résolution appliquée sur les clauses $q(x, b) \vee p(x, a)$ et $r(z, y) \vee \rceil q(f(z), y)$ donnera:
$p(f(z), a) \vee r(z, b)$,
avec comme substitution $\sigma =< x/f(z), y/b >$.

La factorisation de la clause $p(b,y) \vee q(x,f(x),y) \vee p(x,a)$ donnera:
$p(b,a) \vee q(b,f(b),a)$,
avec comme substitution $\sigma =< x/b, y/a >$.

- x=f(z) est unifiable avec succés puisqu'il suffit de remplacer x par f(z) (i.e.,$< x/f(z) >$).
- a=y est unifiable avec succés puisqu'il suffit de remplacer y par a (i.e.,$< y/a >$).
- En conclusion la substitution sera: $\sigma =< x/f(z), y/a >$

2.5.3 Notion de réfutation

Un ensemble de formules est réfutable ssi l'application répétitive des règles de résolution/factorisation entrainera la clause vide $\square$.

2.5.4 propriétés de la résolution

Soient δ un ensemble de clauses et A une clause.

- La résolution est correcte, i.e., si $\delta \vdash A$, alors $\delta \models A$.
- La résolution est complete pour la réfutation, i.e., si δ est insatisfaisable, alors $\delta \vdash \square$,

Remarque
Il existe d'autres systèmes (méthodes) de preuves complets et corrects tels que le calcul de séquents [9]. Ce calcul a été développé par Gerhard Gentzen en 1934; dans cette théorie, chaque ligne de la démonstration (nommée séquent) est une tautologie conditionnelle, i.e., une paire de listes finies de formules propositionnelles. Ce système est largement employé dans la théorie de la preuve. Le lecteur intéréssé trouvera plus de détail sur ce syStème dans les références [9,29].

2.6 Propriétés fondamentales

- La logique des prédicats est correcte, i.e., tout ce qui démontrable est valide (si $\vdash A$, alors $\models A$).
- La logique des prédicats est complete, i.e., tout ce qui valide est démontrable (si $\models A$, alors $\vdash A$.).
- La logique des propositions est **semi-décidable**, parce qu'il n y a pas de procédures effectives qui permettent de dire en un temps fini si une formule F de LPRED est valide ou non.

Remarque
Etant donné une formule F de LPRED, la méthode de résolution nous donnent les scénarios d'exécution suivants:

- Si F est insatisaisable, alors l'algorithme s'arrête en indiquant qu'elle insatisaisable (si F est valide il suffit de considérer $\neg F$ comme entrée).
- L'algorithme peut s'arrêter ou faire une boucle infinie si F est satisfaisable.

2.7 Conclusion

Nous avons présenté dans ce chapitre le calcul des prédicats, cette logique est complete et correcte mais elle est semi-decidable, i.e, il y a des formules satisfaisables que nous ne pouvons pas montrer leur statut de satisfaisabilité. Malgré ce résultat d'indécidabilité, les systèmes de preuve de la logique des prédicats (tels que la résolution) sont largement employés dans la vérification formelle des programmes (citons par exemple la méthode B[5]).

2.8 Exercices

2.8.1 Exercice 1

Considérons les hypothèses suivantes:

- Pour chaque crime il y a quelqu'un qui l' a commis.
- Les gens qui commettent des crimes sont malhonnêtes.
- Les gens arrêtés sont malhonnêtes.
- Pour chaque x, pour chaque y, si x est un malhonnête arrêté et si y est un crime alors x ne peut commettre y.
- Il y a des crimes.

Démontrer qu'il y a des gens malhonnêtes non arrêtés. On vous donne les prédicats suivants : crime(x), commettre(x1,x2), arrêté(x), malhonnête(x).

2.8.2 Exercice 2

On considére les expressions suivantes:

- $H_1 : \forall x \exists z \forall y (A(x, z, y) \rightarrow B(x, y, z))$.
- $H_2 : \forall x \exists y \forall z ((B(x, y, z) \vee C(y)) \rightarrow D(x, z))$.
- $H_3 : \forall x \forall y \forall z A(x, z, y)$.
- $H_4 : \exists x \exists y D(x, y)$.

Est ce que la formule H_4 est déductible à partir des hypothèses H_1, H_2, H_3 (utiliser la méthode de résolution)?

2.8.3 Exercice 3

On considére les expressions suivantes:

- $H_1 : \forall x \forall y ((p(x, y) \vee r(x, a)) \rightarrow q(x))$.
- $H_2 : \exists x \exists y (r(y, x) \rightarrow (p(z, x) \wedge q(y)))$.

[5] https://www.methode-b.com/

1. Donner une interprétation I telle que H1 est vraie ?
2. Donner une interprétation I' telle que H2 est fausse ?
3. Est ce que H2 est insatisfaisable ?

2.8.4 Exercice 4

Formaliser les expréssions suivantes en logique des prédicats:

- Il y a un pays qui est en frontière avec l'algerie et l'égypt.
- a est le plus grand nombre entier.
- Il y a pas d'étudiants qui ont le même numéro de securité sociale.
- Un grandpere est un parent d'un autre parent.
- Seulement les hommes sont mortels.

2.8.5 Exercice 5

Unifier les expressions suivantes:

- $C(a, y, y) = C(z, f(x), f(g(z)))$.
- $P(f(y), z, y) = P(z, f(g(a), g(b))$.

2.8.6 Exercice 6 (QCM)

Choisir la réponse correcte:

- La logique des prédicats sans les symboles fonctionnels est décidable[V/F]?
- Dire si une formule de LPRED est contingente ou non est décidable[V/F]?
- L'algorithme d'unification peut faire une boucle infinie [V/F]?
- Une formule F de LPRED et sa forme clausale sont équivalentes [V/F]?

2.8.7 Exercice 7 (sans corrigé)

On considére les expressions suivantes:

- $H_1 : \forall x \exists y \forall z[(A(x, y) \wedge B(y, z)) \rightarrow \neg(\forall z C(z, x, y))$.
- $H_2 : \forall x \forall z \exists y(A(x, z) \rightarrow B(z, y))$.
- $H_3 : \forall x \forall y \forall z C(x, z, y) \rightarrow D(x, y))$.
- $H_4 : \forall x \forall y(D(x, y) \rightarrow A(x, y))$.
- $H_5 : \exists x \exists y \exists z \neg C(x, z, y)$.

Est ce que la formule H_5 est déductible à partir des hypothèses H_1, H_2, H_3, H_4 (utiliser la méthode de résolution)?

2.8.8 Exercice 8 (sans corrigé)

Soit la formule F1 définie comme suit :
F1$\equiv \forall x \exists w \forall y (r(w, g(x, y)) \rightarrow (r(w, x) \wedge q(w, y))$.

- Trouver une interprétation I telle que F1 est vraire.
- Trouver une interprétation I' telle que F1 est fausse.
- Que peut-on déduire?

Chapitre 3

Introduction à la calculabilité

L'objectif de ce chapitre est de caractériser les problèmes décidables, i.e., ceux qui ont une solution algorithmique (ou ils peuvent être programmés sur une machine) et ceux qui ne l'ont pas. Grâce au concept de Machine de Turing, nous formalisons la notion de procédure effective ou algorithme. Le chapitre de calculabilité (ou théorie de récursion) montre aussi les classes de difficulté relatives aux problèmes de décision (i.e., le temps/éspace nécessaire pour les résoudre).

3.1 Concepts de base

Définition 34 (Alphabet—Problème de décision—Algorithme)

1. *Un alphabet noté Σ est un ensemble fini de symboles (e.g., 0,1,...,9, a,b,c,...). L'ensemble de tous les mots*[1] *(de taille finie) formé sur Σ est noté Σ^*.*
2. *Un problème de décision P est une fonction définie sur un ensemble d'instances (ou mots) I et ayant exactement deux sorties **oui** ou **non** ($P : I \rightarrow \{oui, non\}$). L'ensemble d'entrées I^+ (avec $I^+ \subseteq I$) pour lesquelles la sortie est oui est appelé ensemble d'instances positives et c'est le langage formel (noté L(P)) qui correspond au problème de décision (Donc: $L(P) = \{w/w \in I^+\}$.*
3. *Une procédure effective ou Algorithme: c'est une suite finie d'instructions qui s'arrête pour toute entrée possible $w \in \Sigma$ en donnant un résultat.*

Exemple 3.1.

- **Alphabet:** Si $\Sigma = \{a, b\}$, alors $\Sigma^* = \{\epsilon, a, b, aa, ab, bb, aaa, ...\}$
- **Problème de décision:**
  - P1: Vérifier si un entier n est premier ou non.

[1] Le mot est aussi appelé instance, chaine, ou entrée

- P2: Vérifier si une formule de LPRED est valide ou non.
- P3: Vérifier si deux programmes sont équivalents ou non.

Définition 35 (Machine de Turing) *Une machine de Turing (MT) [30] est un octuplet* $(\Sigma, \Gamma, \square, Q, q_0, q_a, q_r, \delta)$ *où :*

- Q *est l'ensemble des états de l'automate (conventionnellement* Q=$\{q_0, q_1, ..., q_k, q_a, q_r\}$*).*
- q_0 *est l'état initial.*
- q_a *est l'état final d'acceptation.*
- q_r *est l'état final de rejet.*
- Σ *est l'alphabet d'entrée : il permet d'inscrire le mot initial sur le ruban.*
- Γ *est l'alphabet de sortie, c'est l'ensemble des symboles que peut ecrire la machine sur le ruban.*
- $\square$ $(\square \in \Gamma)$ *est le symbole "blanc" qui indique une case vide.*
- $(\delta : (Q - \{qa, qr\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ *est la fonction de transition (le programme de controle de l'automate).*

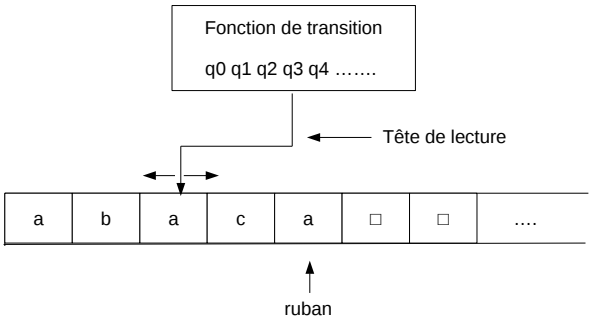

Figure 3.1: Machine de Turing

Initialement, la tête de lecture est positionnée sur le premier symbole de la chaine d'entrée (qui est inscrite sur le ruban), toutes les cases qui suivent la chaine d'entrée contiennent le symbole $\square$. À chaque étape, la MT lit le symbole courant (sous la tête de lecture) et en fonction de son état q_i($q_i \in Q$), la fonction de transition δ (et en particulier la règle $(q_i, x, y, L|R)$ fournit le nouveau symbole

(qui apprtient à Γ) à écrire sur le ruban (i.e., y), la MT change éventuellement son état et se déplace d'une case soit à gauche ou à droite (selon le symbole L ou R de δ).
La MT **s'arrête:**

- Lorsqu'elle atteint un état final q_a.
- Ou q_r.
- Ou lorsque aucune transition ne peut être appliquée.
- Ou la fonction de transition impose un déplacement à gauche au début du ruban.

Le résultat du calcul est toute la chaine qui est écrite sur le ruban à ce moment-là. La chaine d'entrée est acceptée si l'état terminal est q_a. Dans le cas contraire (i.e., la chaine n'est pas acceptée), soit le mot est rejeté parce-que la MT attient l'état q_r , soit la machine boucle indéfinement.

Remarques

- Notre définition de la MT assume que le ruban est borné du coté droit et non borné du coté gauche (voir la figure 3.1).
- Le ruban représente une mémoire de taille infinie.
- Le premier symbole du mot d'entrée est placé dans la première case du ruban, et la fin du mot est délimitée par le premier symbole □ (la suite des symboles □ est infinie).
- La configuration d'une MT est un triplet (q_i, α, β) défini comme suit:
  - $q_i \in Q$ est l'etat de la machine.
  - α est le mot (du ruban) situé avant (à gauche de) la tête de lecture.
  - β est le mot (du ruban) situé aprés (à droite de) la tête de lecture, ce mot contient aussi le symbole courant.
- La MT s'arrête lorsque'elle atteint un état final q_a ou q_r ou aucune transition ne peut être appliquée.
- Un calcul est une suite de configurations qui commence avec (q_0, ϵ, mot) et qui se termine avec une configuration contenant un état final.
- Le déplacement à gauche au début du rubant donne la même position de la tête de lecture.
- L'ensemble des mots pour lesquels une machine de Turing notée M finit avec l'état final q_a est appelé **langage accepté** (reconnu) par M (noté L(M)).

Exemple 3.2.
Sachant que $\sigma = \{a, b\}$, on vous demande de spécifier une MT qui permet de reconnaitre (accepter) les mots de la forme *ab*. Simuler la machine sur la chaine babb.

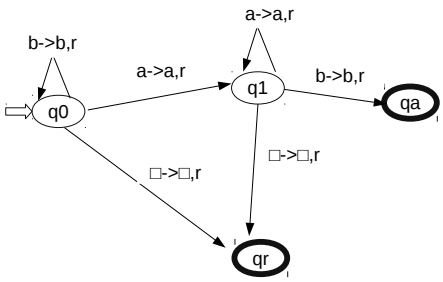

Figure 3.2: MT qui accepte les mots de la forme *ab*

$(q_0, \epsilon, babb) \vdash (q_0, b, abb) \vdash (q_1, ba, bb) \vdash (q_a, bab, b)$.
Il existe plusieurs variantes de la machine de Turing, mais elles possèdent tous le même pouvoir de calcul de la MT standard (i.e., tous ce qui est calculé ou décidé par ces variantes est calculé (ou décidé) par la MT standard). Dans ce qui suit, nous listons quelques variantes:

- Le ruban de la MT est infini dans les deux directions.
- MT avec plusieurs rubans.
- La tête de lecture de la MT peut être stationnaire.
- Le ruban de la MT est bidimensionnel.
- MT non-déterministe (i.e., en partant du même état et du même symbole à lire, la fonction de transition donnera deux résultats (états) différents).
- MT avec accès aléatoire (random access machine).

Définition 36 (Langage décidable) *Un langage (ou problème de décision) est décidable (ou récursif) s'il existe une MT qui accepte tout mot du langage (les instances positives) et qui rejette tout mot n'appartenant pas au langage (les instances négatives). En d'autres termes, elle s'arrête toujours.*

Définition 37 (Langage semi-décidable (ou r.e)) *Un langage (ou problème de décision) est semi-décidable ou récursivement énumérable (r.e) ou reconnaissable s'il existe une MT qui accepte tous les mots du langage (les instances positives) et qui rejette ou boucle indéfinement pour les mots n'appartenant pas au langage (les instances négatives).*

Définition 38 (Langage indécidable) *Un langage (ou problème de décision) est indécidable, (non récursif ou non décidable) s'il n'y a pas de MT qui s'arrête pour tous les mots formés sur l'alphabet.(i.e., soit le langage n'est pas recursivement énumerable, ou le langage est recursivement énumerable mais la machine ne s'arrête pas pour certains mots).*

Définition 39 (Fonction calculable) *Une machine de Turing calcule une fonction $f : \Sigma^* \rightarrow \Gamma^*$ si pour tout mot d'entrée x, elle s'arrête dans une configuration finale où f(x) est ecrit sur le ruban. Une fonction est calculable par une machine de Turing s'il existe une machine de Turing qui la calcule. (La calculabilité est la généralisation de la décidabilité pour les problèmes non binaires).*

La figure 3.3 montre la hiérarchie des langages (problèmes de décision) selon leurs degré de difficulté. Les langages les plus faciles sont les langagaes réguliers reconnus par les automates d'états finis, et les langages les plus difficiles sont les langages indécidables. En 1936, Turing a eu l'idée que n'importe quel programme peut être traité comme une donnée et peut être manipulé par un autre programme (similairement aux compilateurs et interpréteurs actuels qui peuvent manimpuler d'autres programmes). Par conséquent, il a développé la notion de **machine de Turing universelle** qui manipule d'autres programmes comme des données.

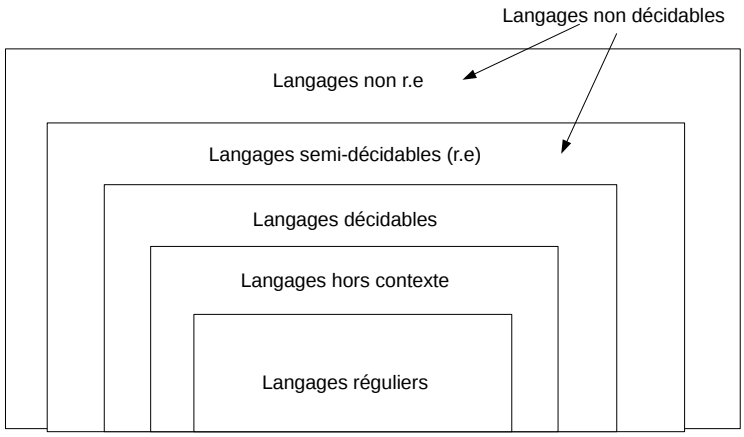

Figure 3.3: Hiérarchie des langages

Définition 40 (Machine de Turing universelle) *La MTU est une machine de Turing particuliere qui prend en entrée une description d'une autre machine de Turing M' et un mot w, ensuite elle simule l'exécution de M' sur w; elle*

retourne "accept" si M' accepte w et elle retourne "reject" si M' rejette w et boucle si M' effectue une boucle infinie.

Nous pouvons facilement se convaincre qu'il existe une machine de Turing M universelle $M_{uinverselle}$ à trois rubans telle que si l'on place :

- la description (codage) $< M >$ de la machine à simuler sur le premier ruban.
- un mot w sur l'alphabet $\Sigma = \{0, 1\}$ sur le second.

alors $M_{uinverselle}$ simule la machine de Turing M sur l'entrée w en utilisant son troisième ruban.

L'importance de la machine de turing universelle, c'est qu'elle peut coder des machines de n'importe quel nombre de symbole d'alphabet et n'importe quel nombre d'états et simule leurs exécutions. Grâce à cette MTU, nous pouvons prouver que certains problèmes (ou langages) sont indécidables.

Remarque

Si un Langage L est reconnaissable (récursivement énumérable) et son complémentaire $\neg L$ est reconnaissable (récursivement énumérable), alors L est décidable.

Exemple 3.3

- Exemples de problèmes décidables:
  - Dire si une formule de la logique des propositions est contingente ou non.
  - Dire si un tableau de N entiers est trié ou non.
  - Dire si un graphe G(X,E) contient un chemin Hamiltonien ou non (i.e., un chemin qui passe par tous les noeuds une et une seule fois).
  - Dire si deux atomes de la logique des prédicats sont unifiables ou non.
  - Dire si un nombre entier est premier ou non.
  - ...
- Exemples de problèmes indécidables:
  - Dire si programme s'arrête ou non lorsqu'il est alimenté avec ses données (semi-décidable ou r.e).
  - Dire si une formule de la logique des prédicats est contingente ou non (non r.e).
  - Dire si une formule de la logique des prédicat est valide ou non (semi-décidable ou r.e).
  - Dire si deux programmes sont équivalents ou non (est ce qu'ils calculent les mêmes sorties).
  - Dire si on peut construire un programme ayant des entrées et des sorties (spécifiés par l'utilisateur) en composant des programmes existants.

– .....

Exemple 3.4 (automates décideurs/reconnaisseurs) Soit l'alphabet $\Sigma = \{a, b\}$ et soit L le langage contenant les mots ayant la forme a*b, spécifier une MT (notée MT1) qui reconnait L (i.e., la MT s'arrête sur les instances positives avec l'état q_a, mais elle peut boucler infiniment sur les instances négatives). Spécifier une autre MT (notée MT2) qui décide le langage L.

MT1=$(\Sigma = \{a, b, \}, \Gamma = \{a, b, \square\}, Q = \{q_0, q_1, q_2, q_a, q_r\}, q_0, q_a, q_r, \delta_1)$, avec δ_1 est montrée dans la figure 3.4 (notons que MT1 fait une boucle infinie pour ϵ, et $\epsilon \notin L$).

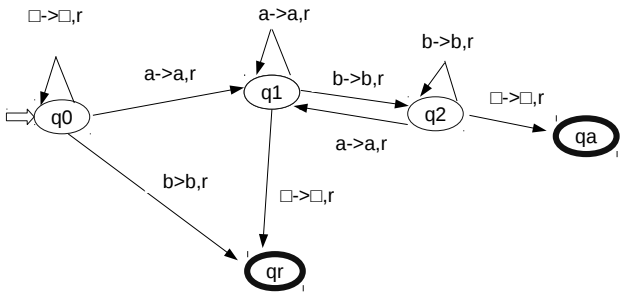

Figure 3.4: MT qui reconnait L

MT2=$(\Sigma = \{a, b, \}, \Gamma = \{a, b, \square\}, Q = \{q_0, q_1, q_2, q_a, q_r\}, q_0, q_a, q_r, \delta_2)$, avec δ_2 est montrée dans la figure 3.5.

Définition 41 (Thèse de Turing-Church) *Les problèmes possèdant une solution algorithmique, sont ceux calculés par une machine de Turing (i.e., ils sont solutionnés par une MT quis'arrête pour toute entrée).*

Remarques

- Tout ce qui ne peut pas être calculable sur une machine de turing n'a aucun algorithme.
- Il y a d'autres modèles de calcul qui sont équivalents au modèle de MT, citons par exemple:
  - Les fonctions du lamda-calcul (Church)[2,3].

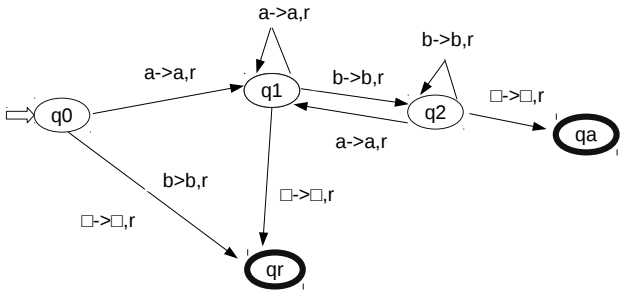

Figure 3.5: MT qui décide L

- Les fonctions partiellement récursives (kleen) [17].
- Les automates cellulaires (Von NeuMann) [31].
- ...

3.2 Indécidabilité du langage diagonal

Dans ce qui suit, nous montrons l'indécidablité du langage diagonal noté D_{TM} (en effet, il est non r.e). Formellement $D_{TM} = \{< M > | < M > \notin L(M)\}$.

Notre langage contient toutes les descriptions des machines de Turing $< M >$ telles que M n'accepte pas sa propore description. Pour faire la démonstration, nous utilisons l'astuce de diagonalisation de Cantor:
Nous construisons une matrice (voir la figure 3.6) contenant les MT (M_i) comme lignes et leurs descriptions $< M_i >$ comme colonnes. La case $(M_j, < M_i >)$ contient 1, si M_j accepte $< M_i >$ et 0 sinon. Nous montrons ensuite l'impossiblité de trouver une MT (notée M_k et son rang est k) telle que $L(M_k) = D_{TM}$, en effet: si la machine M_k existe alors soit $M_k \in L(M_k)$ ou $M_k \notin L(M_k)$:

- Si $M_k \in L(M_k)$ alors par définition de D_{TM} : $M_k \notin L(M_k)$ (contradiction).
- Si $M_k \notin L(M_k)$ alors par définition de D_{TM} : $M_k \in L(M_k)$ (contradiction).

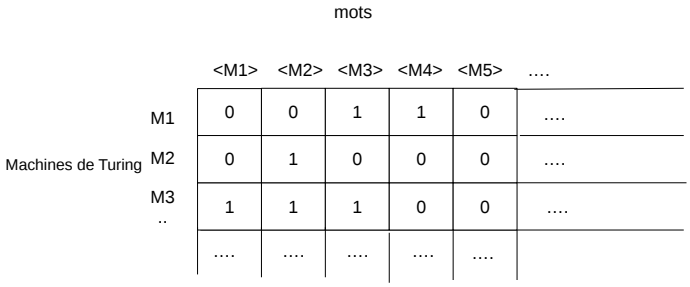

Figure 3.6: Langage diagonal D_{TM}

Conclusion : $M_k \in L(M_k) \leftrightarrow M_k \notin L(M_k)$ (formule toujours fausse). Donc M_k ne peut exister et D_{TM} est indécidable. Dans la suite, on vous demande de faire des réductions à partir de D_{TM} pour montrer que d'autres langages ne sont pas décidables.

3.3 Réduction

Le mécanisme de réduction permet de réutiliser une solution d'un problème déjà existant et l'adapter pour un nouveau problème (e.g. nous pouvons reutiliser le programme qui calcule la superficie d'un rectangle et l'adapter pour calculer la superficie d'un carré). La réduction permet aussi de montrer que certains problèmes sont indécidables.

Définition 42 (Fonction de réduction) *Un langage (problème) A est réductible au langage (problème) B (on note $A \preceq_m B$)[2] (voir la figure 3.7), s'il existe une fonction calculable totale f de Σ^* vers Σ'^* telle que: $\forall w \in \Sigma^* : w \in A \leftrightarrow f(w) \in B$. La réduction f n'a pas besoin d'être bijective (voir l'exemple 3.5 qui montre une réduction valide). L'idée de f est qu'elle associe à chaque mot w qui apprtient à A, un autre mot w' qui appartient à B et à chaque mot qui n'appartient pas à A un autre mot qui n'appartient pas à B.*

En résumé, pour réduire un problème A1 en A2:

- Transformer les inputs de A1 en inputs de A2.
- Simuler l'algorithme Al2 (qui résout A2) sur les entrées calculées précédemment.

[2]L'indice m veut dire many to one

- Interpréter les sorties de Al2 comme étant des réponses pour A1.

Remarques

- Si on a $A \preceq_m B$, alors le problème B est au moins aussi dur que A.
- Si on a $A \preceq_m B$ et B est décidable, alors A est décidable.
- Si on a $A \preceq_m B$ et B est reconnaissable, alors A est reconnaissable.
- Si on a $A \preceq_m B$ et A est indécidable, alors B est indécidable.
- La relation de réduction $\preceq_m$ est réflexive et transitive.

Exemple 3.5 (réductions) Pour réaliser une réduction depuis un problème A vers un problème B, nous initialisons l' entrée de B (notée w') en fonction de l'entrée de A (notée w) ou à l'aides des heuristiques, ensuite il faut que D_B accepte w' ssi D_A accepte w (avec $D_A(.)$, $D_B(.)$ sont les décideurs -s'ils existent- des problèmes A et B réspectivement).
Maintenant, on vous demande de :

- Montrer que $A_{TM} = \{(< M >, x)/$ M accepte x $\}$ est indécidable en faisant cette réduction: $D_{TM} \preceq_m A_{TM}$ ($< M >$ représente le mot qui décrit la machine M).
- Montrer que $HALT_{TM} = \{(< M >, x)/$ M s'arrete sur x $\}$ est indécidable en faisant cette réduction: $A_{TM} \preceq_m HALT_{TM}$
- réduire la fonction qui détermine le plus long chemin dans un graphe vers la fonction qui détermine le plus court chemin d'un graphe.

Question1
Supposons que la MT M est le décideur de A_{TM}, et construisons un décideur S pour D_{TM} comme suit:

Algorithme 4 : S

Input : $< N >$;
1 Run M on input $<< N >, N >$;
2 If M rejects $<< N >, N >$ then accept;
3 If M accepts $<< N >, N >$ then reject;

Observons que L(S) = D_{TM}, mais D_{TM} n'est pas décidable → contradiction (A_{TM} n'est pas décidable).
Question2
Supposons que la MT R est le décideur de $HALT_{TM}$, et construisons un décideur B pour A_{TM} (voir l'algorithme 5).
Question3
Supposons que la MT PCC est le décideur du problème de détermination du plus court chemin d'un graphe représenté par la matrice d'adjacence MA_{n*n} où MA(i,j) donne le coût de l'arc (i,j) et 0 si l'arc n'existe pas. Construisons un décideur PLC pour le problème de détermination du plus long chemin:

Algorithme 5 : B

Input : $(< M >, w)$;
simulate R on $(< M >, w)$;
  If R rejects, (i.e., M loops) then reject;
  If R accepts, (i.e., M halts) then simulate M(w);
    If M accepts then accept;
    If M rejects then reject;

PCC possède la signature suivante :
input:MA_{n*n}: graph; a,b $\in Nodes(MA_{n*n})$ (i.e., le noeud source et le noeud destination)
output: (path $\in ListOfNodes$, cost : real)
Le décideur PLC est établi comme suit:

Algorithme 6 : PLC

Input : MA_{n*n}: graph;
a,b $\in Nodes(MA_{n*n})$; MA'_{n*n}:graph;
for each i $\in Nodes(MA_{n*n})$;
  for each j $\in Nodes(MA_{n*n})$;
  $MA'(i,j) = -1 * MA(i,j)$;
(path',cost')=PCC(MA'_{n*n}, a, b);
return (path',-1*cost');

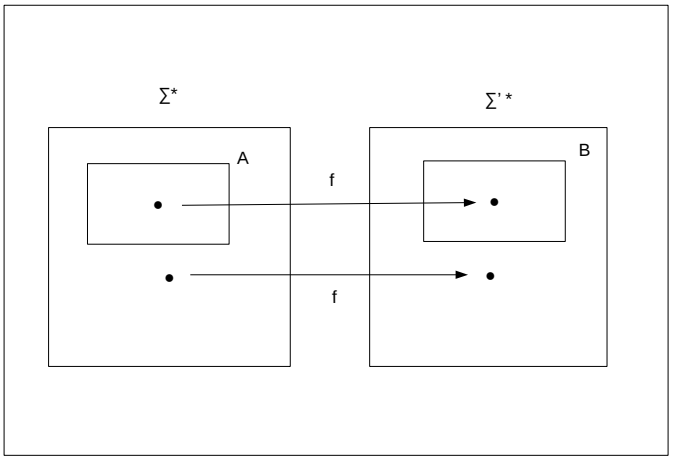

Figure 3.7: Notion de réduction

3.3.1 Théorème de Rice (1951)

Toute propriété P sur les langages reconnus par les MT (r.e) et qui n'est pas triviale est indécidable.

Remarques

- La propriété P porte sur le langage accepté par la MT et non pas sur son implementation (telle que le nombre d'états ou le nombre de symboles d'alphabet). En d'autres mots, P est une propriété d'un langage recursivement énumerable, i.e., accepté par une MT et non pas d'une MT.
- Une propriété P est non-triviale s'il y a deux MT M1 et M2 telles que $L(M1) \in P$ et que $L(M2) \notin P$. En d'autres mots il y a deux langages r.e: L1 et L2 tels que L1 satisfait P et L2 ne satisfait pas P.
- Une propriété P est triviale si elle est tout le temps fausse ou tout le temps vraie (tous les langages r.e vérifient P ou tous les langages r.e ne vérifient pas P).

Exemple 3.6

- Exemples de propriétés relatives aux langages r.e (reconnus par une MT M):
  - L(M) contient ϵ.
  - L(M) est fini.
  - La taille de L(M) est égale 3.
  - M accepte la chaine "0011".
  - L(M) est régulier.
  - L(M)= Σ^*.
  - ...
- Exemples de propriétés relatives à des MT M et pas aux langages (r.e) reconnus par M:
  - M contient contient au moins 50 états.
  - M s'arrête sur tous les mots
  - M s'arrête aprés un maximum de 100 étapes de calcul.
  - M rejette ϵ.
  - M rejette la chaine "0011".
  - Il y a une machine M' qui est plus petite que M et qui est équivalente à M.
  - ...

3.4 Complexité

Le deuxième objectif de la théorie de calculabilité est de mesurer la complexité des problèmes de décision (le temps/éspace nécessaire pour les résoudre). Dans cette optique, les experts ont développé une hiérarchie de problèmes qui part des problèmes P (polynomiaux) jusqu'aux problémes indécidables.

3.4.1 Problèmes polynomiaux (P)

Un problème $p' \in P$ s'il est résolu en faisant au maximum n^k étapes (instructions/transitions de la MT), avec k une constante et n la taille du mot d'entrée.
Exemple 3.7

- Trier un tableau d'entiers de taille n.
- La recherche du plus court chemin entre deux points (nœuds) d'un graphe.
- Dire si une formule 2-FNC est satisfaisable ou non.

3.4.2 Problèmes non-déterministes polynomiaux (NP)

Un problème $p' \in NP$ si la vérification d'une solution potentielle nécessite un temps polynomial (le nombre de solutions potentielles peut être exponentiel).

Remarque

- **Problème NP-Hard:** Un Problème A est dit NP-Hard ssi:
  - $\forall B \in NP : B \preceq_m A$ (Un problème NP-Hard peut appartenir à des classes de complexité plus élevées que NP et même indecidables).
- **Problème NP-Complet:** Un Problème A est dit NP-Complet ssi:
  - A $\in NP$.
  - $\forall B \in NP : B \preceq_m A$ (i.e., sa difficulté est supérieure ou égale à la difficulté de tous les problèmes NP, en d'autres mots, si on lui trouve une solution polynomiale, alors tous les problèmes NP ont aussi une solution polynomiale)

3.4.3 P Vs. NP

La question d'égalité entre les classes P et NP (i.e., P≡ NP), n'est pas solutionnée jusqu'à maintenant, bienque la majorité des chercheurs pensent que les deux classes sont différentes, il n' y a pas de preuve établie jusu'à maintenant. L'institut de mathématique de Clay en USA donnera un prix d'un million de dollars pour la personne qui résoudra cette question. P Vs. NP consiste à trancher s'il y a une solution algorithmique polynomiale à l'un des problèmes NP-Complets; pour ce faire, il faut qu'on résout un problème de recherche sans faire la recherche !
Exemple 3.8
Dans ce qui suit, nous citons quelques problèmes fameux de la classe NP.

- Problème du voyageur de commerce (PVC) (NP-complet).
- 3-SAT, 4-SAT (NP-complet).
- 2-partition : étant donné n entiers, peut-on les diviser en deux groupes dont la somme des élements est la même? (NP-complet)

- 3-partition (NP-complet).
- K-coloriage de graphe (NP-complet).
- Sudoku (NP-complet) .
- Knapsack (ou sac à dos) (NP-complet).
- La recherche d'une clique de taille k dans un graphe G(X,E) (NP-complet).
- Factorisation d'un nombre entier (NP).
- ...

3.4.4 Problèmes polynomiaux en termes d'espace (PSPACE)

Un problème $p' \in PSPACE$ s'il est résolu en utilisant au maximum $O(n^k)$ case-mémoires, avec k une constante et n la taille du mot d'entrée (en d'autres mots, la MT visite au maximum $O(n^k)$ cellules du ruban).

En prenant l'exemple des formules booléennes quantifiées (QBF) (qui est l'un des problèmes les pus difficiles de la classe PSPACE ou PSPACE-Complet). Ce problème consiste à dire si la formule F est vraie:
$F \equiv \exists x_1 \forall x_2, ...\exists x_{n-1} \forall x_n \phi(x_1, ..x_n)$. Notons qu'il y a $\frac{n}{2}$ variables (on suppose que n est pair) ou atomes quantifiées avec $\exists$ et $\frac{n}{2}$ variables ou atomes quantifiées avec $\forall$. Il s'agit de trouver une affectation des variables quantifiées avec $\exists$ de telle sorte que pour toute valeur possible des variables quantifiées avec $\forall$, la formule F est vraie. Il est clair que dans ce cas, la vérification de la correction d'une solution potentielle (une affectation des variables quantifiées avec $\exists$) est exponentielle (à moins que P=NP, et dans ce cas, on peut vérifier la solution en temps polynomial).
Exemple 3.9

- Formules booléennes quantifiées (QBF).
- Jeux de géographie: en considérant un graphe G(X,E), le jeux (à deux adversaires) invite le joueur A à choisir un noeud du graphe G, ensuite le joueur B choisit un autre noeud de G (non déjà sélectionné) et qui possède un arc (orienté) avec le noeud précédent, le jeux se répete jusqu'à ce que l'un des joueurs est bloqué (et donc il perd). La question qui se pose est est ce qu'il y a une strategie gagnante pour le joueur A ?
- Problèmes de planification : construire une séquence d'actions $a_1...a_n$ qui mène à une configuration finale E_F à partir d'une configuration initial E_I.
- ...

3.4.5 Problèmes exponentiels (EXP)

Un problème $p' \in EXP$, s'il est résolu en un temps exponentiel, i.e, la MT fait $O(2^{n^k})$ étapes pour achever le problème. (avec k une constante et n la taille du mot d'entrée).

Il est prouvé que cette classe contient des problèmes qui n'appartiennent pas à P (l'inclusion entre EXP et P est stricte); par exemple nous pouvons

citer le problème d'arrêt borné d'une MT (BHM) comme membre à cette classe: considérons une MT M et son entrée x, le problème BHM consiste à vérifier si M s'arrête sur x après k transitions ou moins (k est codé sur $\{0,1\}$ et occupe t cellules sur le ruban)?
Ce problème a une complexité exponentielle puisqu'en pire des cas, on doit simuler M pour les 2^t étapes possibles, et à la fin on tranche si la paire $(< M >, x)$ est acceptée ou non.
Exemple 3.10
Nous citons par la suite quelques problèmes appartenant à **EXP**:

- Jeu d'échec (généralisé, i.e, avec un échiquier de taille n*n): Décider si le joueur A réussit ou non une épreuve d'échec, en partant d'une configuration particulière de l'échiquier.
- Jeu de dame (géneralisé, i.e, avec un échiquier n*n).
- Le problème d'arrêt borné d'une MT (Bounded Halting Machine).

3.4.6 Problèmes recursifs (R)

Un problème $p' \in R$, s'il est résolu en un temps fini, i.e, la MT exécute un nombre fini de transitions avant de s'arrêter. (la classe R coincide avec les problèmes decidables et englobe les classes de complexité précédentes).

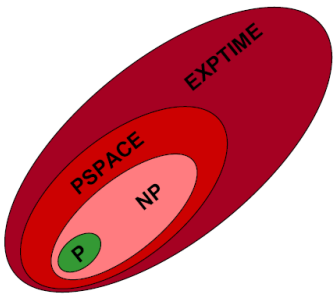

Figure 3.8: Une partie de la hiérarchie des classes de complexité

3.5 Conclusion

Nous avons montré dans ce chapitre que la majorité des problèmes existants sont indécidables (ils n'ont pas de solution algorithmique), et la minorité qui reste est décidable mais elle a plusieurs niveaux de complexité (allant de la classe P jusqu'à la classe R). Bien que l'indécidablité des problèmes est presque une règle (et non pas une excéption), nous pouvons toujours rechercher des sous-cas ou des sous-problèmes (du problème original indécidable) qui soient décidables ou à la limite semi-décidables; en pratiquant cette philosophie, nous pouvons apporter des remèdes à beaucoup de problèmes courants.

3.6 Exercices

3.6.1 Exercice 1

Créer la MT qui décide le langage contenant les mots ayant la forme $w\#w^r$ (w^r est le mirroir de w), avec $w \in \{a, b\}^*$.

3.6.2 Exercice 2

Sachant que $\Sigma = \{a, b\}$, on vous demande de créer la MT qui décide les mots ayant la forme w#w' avec w,w' $\in \{0, 1\}^*$ et le nombre de 0 de w est égal au nombre de 1 de w'.

3.6.3 Exercice 3

Sachant que $\Sigma = \{a, b, c\}$, on vous demande de créer la MT qui décide les mots ayant la forme a*a ou b*b.

3.6.4 Exercice 4 (sans corrigé)

Sachant que $\Sigma = \{a, b\}$, on vous demande de créer la MT qui décide les mots w $\in \{a, b\}^*$ et qui ont autant de a que de b.

3.6.5 Exercice 5

Supposons que la MT M1 reconnait L et la MT M2 reconnait $\neg L$, et soit la MT S:

Algorithme 7 : S

Input : x
1 If M1(x) accept then accept;
2 If M2(x) accept then reject;

Dites pourquoi S n'est pas un décideur de L ?

- **(a)** S rejette des mots appartenant à L.
- **(b)** M2 peut faire une boucle infinie.
- **(c)** M1 peut faire une boucle infinie.
- **(d)** M2 peut faire une boucle infinie.
- **(e)** S accepte des mots n'appartenant pas à L.

3.6.6 Exercice 6

Soit le langage $L = \{< M > /$ M s'accepte rien $\}$, on vous demande de choisir la (les) bonne(s) réponses:

- **(a)** L est reconnaissable.

- **(b)** L est n'est pas reconnaissable.
- **(c)** $\neg L$ est reconnaissable.
- **(d)** $\neg L$ n'est pas reconnaissable.

3.6.7 Exercice 7

Montrer que le langage $E_{TM} = \{< M > | L(M) = \emptyset\}$ n'est pas décidable, en faisant une réduction à partir de A_{TM}.

3.6.8 Exercice 8

Montrer que le langage (Equivalence) $EQ_{TM} = \{< M1, M2 > | L(M1) = L(M2)\}$ n'est pas décidable, en faisant une réduction à partir de E_{TM}.

3.6.9 Exercice 9 (sans corrigé)

Montrer que le langage $REG_{TM} = \{< M > | L(M) estregulier)\}$ n'est pas décidable, en faisant une réduction à partir de $HALT_{TM}$.

3.6.10 Exercice 10 (sans corrigé)

Montrer que le langage $REV_{TM} = \{< M > | w \in L(M) ssimirroir(w) \in L(M))\}$ n'est pas décidable, en faisant une réduction à partir de $HALT_{TM}$.

3.6.11 Exercice 11

réduire une formule SAT en format FNC en une formule 3-FNC?

3.6.12 Exercice 12 (sans corrigé)

réduire une formule SAT en format 3-FNC en une formule 2-FNC?

Annexes

.1 Annexe A: Logique propositionnelle

.1.1 Exercice 1

$\models (A \leftrightarrow B) \leftrightarrow ((B \leftrightarrow C) \leftrightarrow (A \leftrightarrow C))$ La table de vérité suivante montre que la conclusion est valide.

A	B	C	$A \leftrightarrow B(1)$	$B \leftrightarrow C$ (2)	$A \leftrightarrow C(3)$	(2) $\leftrightarrow$ (3)	(1) $\leftrightarrow$ ((2) $\leftrightarrow$ (3))
1	1	1	1	1	1	1	1
0	1	1	0	1	0	0	1
1	0	1	0	0	1	0	1
0	0	1	1	0	0	1	1
1	1	0	1	0	0	1	1
0	1	0	0	0	1	0	1
1	0	0	0	1	0	0	1
0	0	0	1	1	1	1	1

.

La table de vérité suivante montre que la conclusion est une conséquence sémantique des prémisses.
$\{D \rightarrow B, D \rightarrow F, \rceil(B \wedge F), (E \vee D)\} \models E$.
$D \rightarrow B$ est notée (1), $D \rightarrow F$ est notée (2), $\rceil(B \wedge F)$ est notée (3),$(E \vee D)$ est notée (4).

D	B	F	E	(1)	(2)	(3)	(4)	(1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4))	E
1	1	1	1	1	1	0	1	0	1
0	1	1	1	1	1	0	1	0	1
1	0	1	1	0	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1
1	1	0	1	1	0	1	1	0	1
0	1	0	1	1	1	1	1	1	1
1	0	0	1	0	0	1	1	0	1
0	0	0	1	1	1	1	1	1	1
1	1	1	0	1	1	0	1	0	0
0	1	1	0	1	1	1	0	0	0
1	0	1	0	0	1	1	1	0	0
0	0	1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1	0	0
0	1	0	0	1	1	1	0	0	0
1	0	0	0	0	0	1	1	0	0
0	0	0	0	1	1	1	0	0	0

.1.2 Exercice 2

Démonstration par résolution (Question 1):

$[(A \wedge C) \rightarrow F) \wedge ((D \wedge \rceil E) \rightarrow F) \wedge (A \rightarrow (C \vee D)) \wedge (F \rightarrow E) \wedge (A \rightarrow \rceil E)] \vdash \rceil A$.

La forme FNC des prémisses en plus de la négation de la conclusion sont données comme suit:

- H1: $(\rceil A \vee \rceil C \vee F)$
- H2: $(\rceil D \vee E \vee F)$
- H3: $(\rceil A \vee C \vee D)$

- H4: $(\rceil F \vee E)$
- H5: $(\rceil A \vee \rceil E)$
- H6: A

résolution:

- RR appliquée sur H5 et H6 donne H7: $(\rceil E)$
- RR sur H7 et H4 donne H8: $(\rceil F)$
- RR sur H8 et H2 donne H9: $(\rceil D \vee E)$
- RR sur H9 et H7 donne H10: $(\rceil D)$
- RR sur H8 et H1 donne H11: $(\rceil A \vee \rceil C)$
- RR sur H11 et H6 donne H12: $(\rceil C)$
- RR sur H12 et H3 donne H13: $(\rceil A \vee D)$
- RR sur H13 et H10 donne H14: $(\rceil A)$
- RR sur H14 et H6 donne H15: $\square$

Démonstration par résolution (Question 2):
$[((B \wedge F) \rightarrow (A \vee E)) \wedge (\rceil C \rightarrow (E \vee F)) \wedge ((B \wedge A) \rightarrow C) \wedge (E \rightarrow \rceil B) \wedge (C \rightarrow D) \wedge B] \vdash D$.
La forme FNC des prémisses en plus de la négation de la conclusion sont données comme suit:

- H1: $(\rceil B \vee \rceil F \vee A \vee E)$
- H2: $(C \vee E \vee F)$
- H3: $(\rceil B \vee \rceil A \vee C)$
- H4: $(\rceil E \vee \rceil B)$
- H5: $(\rceil C \vee D)$
- H6: B
- H7: $\rceil D$

résolution:

- RR appliquée sur H5 et H7 donne H8: $(\rceil C)$
- RR sur H6 et H4 donne H9: $(\rceil E)$
- RR sur H9 et H2 donne H10: $(C \vee F)$
- RR sur H10 et H8 donne H11: (F)
- RR sur H3 et H6 donne H12: $(\rceil A \vee C)$
- RR sur H12 et H8 donne H13: $(\rceil A)$

- RR sur H13 et H1 donne H14: $(\rceil B \vee \rceil F \vee E)$
- RR sur H14 et H9 donne H15: $(\rceil B \vee \rceil F)$
- RR sur H15 et H11 donne H16: $(\rceil B)$
- RR sur H16 et H6 donne H17: $\square$

.1.3 Exercice 3

Nous utilisons une variable notée X_y^i pour dire que la wilaya y de la figure 1.6 (y est la première lettre de la capitale de la wilaya) est coloriée avec la couleur i. Le modele en logique des propositions du problème devient:

- pour tout $y \in \{t, n, a, o, s\}$:$(X_y^r \vee X_y^v \vee X_y^b)$. (les 05 wilayas peuvent être coloriés avec le rouge ou le bleu ou le vert).
- pour tout $i \in \{r, v, b\}$:$\neg(X_t^i \wedge X_a^i$ (puisque t et a sont adjacentes, elles doivent pas être coloriées avec la même couleur).
- pour tout $i \in \{r, v, b\}$:$\neg(X_o^i \wedge X_a^i$ (puisque o et a sont adjacentes, elles doivent pas être coloriées avec la même couleur).
- pour tout $i \in \{r, v, b\}$:$\neg(X_s^i \wedge X_a^i$ (puisque s et a sont adjacentes, elles doivent pas être coloriées avec la même couleur).
- pour tout $i \in \{r, v, b\}$:$\neg(X_t^i \wedge X_n^i$ (puisque t et n sont adjacentes, elles doivent pas être coloriées avec la même couleur).
- pour tout $i \in \{r, v, b\}$:$\neg(X_t^i \wedge X_s^i$ (puisque t et s sont adjacentes, elles doivent pas être coloriées avec la même couleur).
- pour tout $i \in \{r, v, b\}$:$\neg(X_s^i \wedge X_o^i$ (puisque s et o sont adjacentes, elles doivent pas être coloriées avec la même couleur).
- pour tout $i \in \{r, v, b\}$:$\neg(X_s^i \wedge X_n^i$ (puisque s et n sont adjacentes, elles doivent pas être coloriées avec la même couleur).

.1.4 Exercice 4

Modélisation des contraintes linéaires en logique des propositions (les 04 variables sont binaires).

- $[X_1 + X_2 + X_3 + X_4 = 1]$ est représentée par la conjonction des deux sous formules:
  - $X_1 \vee X_2 \vee X_3 \vee X_4$ (au moins une variable est vraie).
  - $\wedge_{i=1}^{3} \wedge_{j=i+1}^{4} \neg(X_i \wedge X_j)$ (au plus une variable est vraie, i.e., exclusion mutuelle entre variables).
- $[X_1 + X_2 + X_3 + X_4 < 3]$ est représentée le fait que les élements des C_3^4 sous-ensembles de variables ne doivent pas être vrais en même temps. Ceci est donné par la par la conjonction des sous formules suivantes:

- $\neg(X_1 \wedge X_2 \wedge X_3)$ (les élements X_1, X_2, X_3 ne doivent pas être vrais en même temps).
- $\neg(X_1 \wedge X_2 \wedge X_4)$ (les élements X_1, X_2, X_3 ne doivent pas être vrais en même temps).
- $\neg(X_1 \wedge X_3 \wedge X_4)$ (les élements X_1, X_2, X_3 ne doivent pas être vrais en même temps).
- $\neg(X_2 \wedge X_3 \wedge X_4)$ (les élements X_1, X_2, X_3 ne doivent pas être vrais en même temps).

.1.5 Exercice 7 (QCM)

Les réponses correctes sont:

- **V**
- **V**
- **F**
- **(b),(d)**
- **V**
- **F**
- **(b)**
- **V**

.2 Annexe B: Logique des prédicats

.2.1 Exercice 1

- $H_1 : \forall x \exists y(crime(x) \rightarrow commettre(y, x))$.
- $H_2 : \forall x \forall y((crime(x) \wedge commettre(y, x)) \rightarrow malhonnete(y))$.
- $H_3 : \forall x(arrete(x) \rightarrow malhonnete(x))$.
- $H_4 : \forall x \forall y[(malhonnete(x) \wedge arrete(x) \wedge crime(y)) \rightarrow \neg commettre(x, y)]$.
- $H_5 : \exists y(crime(y))$.
- $Conclusion : \exists x(malhonnete(x) \wedge \neg arrete(x))$.

La négation de la conclusion notée H_6 est : $\forall x(\neg malhonnete(x) \vee arrete(x))$.
La forme clausale de chaque formule H_i est obtenue comme suit:

- $H'_1 : (\neg crime(x_1) \vee commettre(f(x_1), x_1))$.
- $H'_2 : (\neg crime(x_2) \vee \neg commettre(y_2, x_2) \vee malhonnete(y_2))$.
- $H'_3 : (\neg arrete(x_3) \vee malhonnete(x_3))$.
- $H'_4 : [\neg malhonnete(x_4) \vee \neg arrete(x_4) \vee \neg crime(y_4) \vee \neg commettre(x_4, y_4)]$.
- $H'_5 : crime(a)$.
- $H'_6 : (\neg malhonnete(x_6) \vee arrete(x_6))$.

maintenant, il faut prouver que $\{H'_1, H'_2, H'_3, H'_4, H'_5, H'_6\} \vdash \square$

- La règle de résolution (RR) appliquée sur H'_1, H'_5 donne :$H_7 : [commettre(f(a), a)]$, avec la substitution $< x_1/a >$.
- RR appliquée sur H'_4, H'_5 donne :$H_8 : [\neg malhonnete(x_4) \vee \neg arrete(x_4) \vee \neg commettre(x_4, a)]$, avec la substitution $< y_4/a >$.
- RR appliquée sur H_8, H'_6 donne :$H_9 : [\neg malhonnete(x_4) \vee \neg malhonnete(x_4) \vee \neg commettre(x_4, a)]$, avec la substitution $< x_6/x_4 >$.
- La factorisation de H_8 donne :$H_9 : [\neg malhonnete(x_4) \vee \neg commettre(x_4, a)]$.
- RR appliquée sur H_9, H'_2 donne :$H_10 : [\neg crime(x_2) \vee \neg commettre(x_4, x_2) \vee \neg commettre(x_4, a)]$, avec la substitution $< y_2/x_4 >$.
- La factorisation de H_10 donne :$H_11 : [\neg crime(x_2) \vee \neg commettre(x_4, a)]$.
- RR appliquée sur H_10, H'_5 donne :$H_11 : [\neg commettre(x_4, a)]$, avec la substitution $< x_2/a >$.
- RR appliquée sur H_11, H_7 donne :$H_12 : \square$, avec la substitution $< x_4/f(a) >$.

.2.2 Exercice 2

On considére les expressions suivantes:

- $H_1 : \forall x \exists z \forall y (A(x,z,y) \rightarrow B(x,y,z))$.
- $H_2 : \forall x \exists y \forall z ((B(x,y,z) \vee C(y)) \rightarrow D(x,z))$.
- $H_3 : \forall x \forall y \forall z A(x,z,y)$.
- $H_4 : \exists x \exists y D(x,y)$.

Est ce que la formule H_4 est déductible à partir des hypothèses H_1, H_2, H_3 (utiliser la méthode de résolution)? La négation de H_4 notée H_5 est obtenue comme suit:
$H_5 : \forall x \forall y \neg D(x,y)$. La forme clausale de chaque formule H_i est obtenue comme suit:

- $H'_1 : (\neg A(x_1, f(x_1), y_1) \vee B(x_1, y_1, f(x_1)))$.
- H_2 est éclatée en deux sous formules
  - $H'_2 : (\neg B(x_2, g(x_2), z_2) \vee \neg D(x_2, z_2))$.
  - $H'_3 : (\neg B(g(x_2)) \vee \neg D(x_2, z_2))$. Après renommage, elle devient: $H''_3 : (\neg B(g(x'_2)) \vee \neg D(x'_2, z'_2))$.
- $H'_4 : A(x_4, z_4, y_4)$.
- $H'_5 : \neg D(x_5, y_5)$.

Maintenant, il faut prouver que $\{H'_1, H'_2, H''_3, H'_4, H'_5\} \vdash \square$

- La règle de résolution (RR) appliquée sur H'_2, H'_5 donne :$H'_6 : (\neg B(x_2, g(x_2), z_2)$, avec la substitution $< (x_5/x_2), (y_5/z_2) >$.
- RR appliquée sur H'_1, H'_6 donne :$H'_7 : [\neg A(x_2, f(x_1), g(x_2))$, avec la substitution $< (x_1/x_2), (z_2/f(x_1)), (y_1/g(x_2)) >$.
- RR appliquée sur H'_7, H'_4 donne :$\square$, avec la substitution $< (x_4/x_2), (z_4/f(x_1)), (y_4/g(x_2)) >$.

.2.3 Exercice 3

Construisons l'interprétation I pour l'expression H_1:

- $H_1 : \forall x \forall y ((p(x,y) \vee r(x,a)) \rightarrow q(x))$.
- $H_2 : \exists x \exists y (r(y,x) \rightarrow (p(z,x) \wedge q(y)))$.

I est définie comme suit:

- $D = \{1, 2\}$.
- $[[p]] = \{(1,1), (2,2), (2,1), (1,2)\}$, $[[r]] = \{(1,1), (2,2), (2,1), (1,2)\}$, $[[q]] = \{1,2\}$,$[[a]] = \{1\}$. et par conséquent:
  - $[[\forall x \forall y ((p(x,y)]] = 1$.
  - $[[\forall x \forall y ((r(x,a)]] = 1$.

– $[[\forall x q(x)]] = 1$ et donc H_1 est vraie.

Construisons l'interprétation I' pour l'expression H_2:

- $D = \{1, 2\}$.
- $[[p]] = \{(1,1),(2,2),(2,1),(1,2)\}$, $[[r]] = \{(1,1),(2,2),(2,1),(1,2)\}$, $[[q]] = \emptyset$,$[[z]] = \{1\}$. et par conséquent:
  - $[[\exists x \exists y((r(y,x)]] = 1$, puisque si x=1 et y=1, alors $(y,x) \in [[r]]$.
  - $[[\exists y((q(y)]] = 0$, puisque $[[q]] = \emptyset$, et donc $[[\exists x \exists y(p(z,x) \wedge q(x)]] = 0$.
  - Par conséquent H_2 est fausse.

H_2 n'est pas insatisfaisable, mais elle est **contingente**, parce qu'on peut considérer l'interprétation I" dans laquelle H_2 est vraie :

- $D = \{1, 2\}$.
- $[[p]] = \{(1,1),(2,2),(2,1),(1,2)\}$, $[[r]] = \{(1,1),(2,2),(2,1),(1,2)\}$, $[[q]] = \{1,2\}$,$[[z]] = \{1\}$ (il suffit de considérer le cas x=1 , y=1.

.2.4 Exercice 4

Formalisation des expréssions en LPRED:

- Il y a un pays qui est en frontière avec l'algerie et l'égypt.
- $H_1 : \exists x(Pays(x) \wedge EnFrontiere(x, algerie) \wedge EnFrontiere(x, egypt))$
- a est le plus grand nombre entier.
- $H_2 : Entier(a) \wedge (\forall x(Entier(x) \rightarrow (a > x))$.
- Il n'y a pas d'étudiants qui ont le même numéro de securité sociale.
- $H_3 : \forall x \forall y((Etudiant(x) \wedge etudiant(y)) \rightarrow (SecSocial(x) \neq SecSocial(y)))$.
- Un grand-père est un parent d'un autre parent.
- $H_4 : \forall x \forall y[GrandPere(x,y) \rightarrow \exists z(Parent(x,z) \wedge Parent(z,c))]$.
- Seulement les hommes sont mortels.
- $H_5 : \forall x[Mortel(x) \rightarrow Homme(x)]$.

.2.5 Exercice 5

Unification des expressions suivantes:

- $C(a,y,y) = C(z, f(x), f(g(z)))$.
- $P(f(y), z, y) = P(z, f(g(a), g(b))$.

Question 1:$C(a,y,y) = C(z, f(x), f(g(z)))$

- étape1:

- $a = z$ et donc z=a (ok)
- y=f(x)
- y=f(g(z))

- étape 2:
  - z=a (ok)
  - y=f(x) (ok)
  - y=f(g(z))

- étape 3:
  - z=a (ok)
  - y=f(x) (ok)
  - f(x)=f(g(z))

- étape 4:
  - z=a (ok)
  - y=f(x) (ok)
  - x=g(z) (ok)

- étape 5:
  - z=a (ok)
  - y=f(g(a)) (ok)
  - x=g(a) (ok)

Donc, les deux atomes sont unifiables grâce à la substitution $< (z, a), (y, f(g(a))) >$
Question 2: $P(f(y), z, y) = P(z, f(g(a), g(b))$.

- étape1:
  - $f(y) = z$ et donc z=f(y) (ok)
  - z=f(g(a))
  - y=g(b)

- étape 2:
  - z=f(y) (ok)
  - f(y)=f(g(a))
  - y=g(b)

- étape 3:
  - z=f(y) (ok)
  - y=g(a) (ok)
  - y=g(b)

- étape 4:
  - z=f(g(a)) (ok)
  - y=g(a) (ok)
  - g(a)=g(b)
- étape 5:
  - z=f(g(a)) (ok)
  - y=g(a) (ok)
  - a=b (échec → les deux atomes ne sont pas unifiables)

.2.6 Exercice 6(QCM)

Les réponses correctes sont:

- **V**
- **F**
- **F**
- **F**

.3 Annexe C: Introduction à la calculabilité

.3.1 Exercice 1

Le langage L1=$\{w\#w^r\}$ est décidé par la MT suivante:
$M_{w\#w^r}$=$(\Sigma = \{a, b, \#\}, \Gamma = \{a, b, \#, \square\}, Q = \{q_0, q_1, q_2, ..., q_{11}, q_a, q_r\}, q_0, q_a, q_r, \delta)$,
avec δ est montrée dans la figure 9.

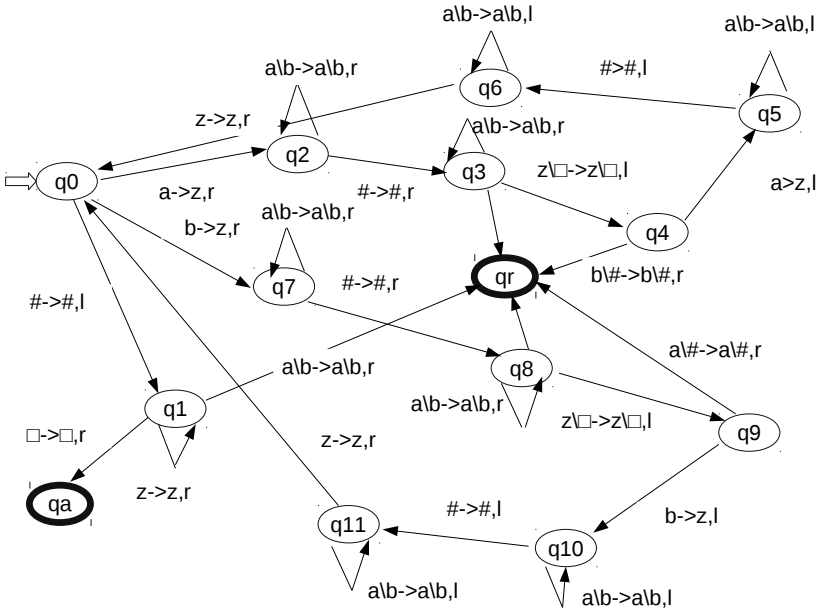

Figure 9: MT qui décide L1

•

.3.2 Exercice 2

Le langage L2=$\{w\#w'\}$, avec le nombre de 0 de w est égal au nombre de 1 de w', est décidé par la MT suivante:
$MT_{w\#w'}$=$(\Sigma = \{0, 1, \#, \}, \Gamma = \{0, 1, \#, \square\}, Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_a, q_r\}, q_0, q_a, q_r, \delta)$,
avec δ est montrée dans la figure 10.

.3.3 Exercice 3

Sachant que $\Sigma = \{a, b, c\}$, on vous demande de créer la MT qui décide . Le langage L3 (les mots ayant la forme a*a ou b*b) est décidé par la MT suivante:
$MT_{a*a|b*b}$=$(\Sigma = \{a, b, c\}, \Gamma = \{a, b, c, \square\}, Q = \{q_0, q_1, q_2, q_3, q_4, q_a, q_r\}, q_0, q_a, q_r, \delta)$,
avec δ est montrée dans la figure 11.

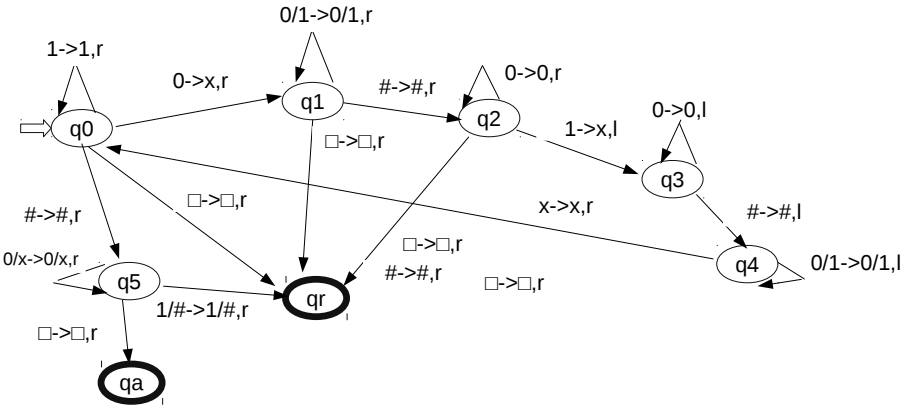

Figure 10: MT qui décide L2

.3.4 Exercice 5

Réponse correcte : **(c)**.

.3.5 Exercice 6

Réponses correctes : **(b),(c)**.

.3.6 Exercice 11

Pour convertir une formule en format FNC en une formule en format 3-FNC, nous proposons un algorithme (X-FNC-TO-3-FNC) qui discutera les 03 cas suivants:
(r1): La clause C_i possède un seul littéral (lignes 5-9 de l'algorithme X-FNC-TO-3-FNC): dans ce cas, nous ajoutons deux variables $x_1, x2$ avec les 04 possibiltés des valeurs de vérité (i.e., 04 nouvelles clauses) afin de garder l'équivalence avec la clause originale c_i.
(r2): La clause C_i possède deux littéraux (lignes 10-13de l'algorithme X-FNC-TO-3-FNC): dans ce cas, nous ajoutons une variable x_1 avec ses 02 valeurs de vérité (i.e., 02 nouvelles clauses) afin de garder l'équivalence avec la clause originale c_i.
(r3): La clause C_i possède m littéraux (m¿ 03, voir les lignes 18-24 de l'algorithme X-FNC-TO-3-FNC): dans ce cas, nous créons un nouvel atome r qui est équivalent

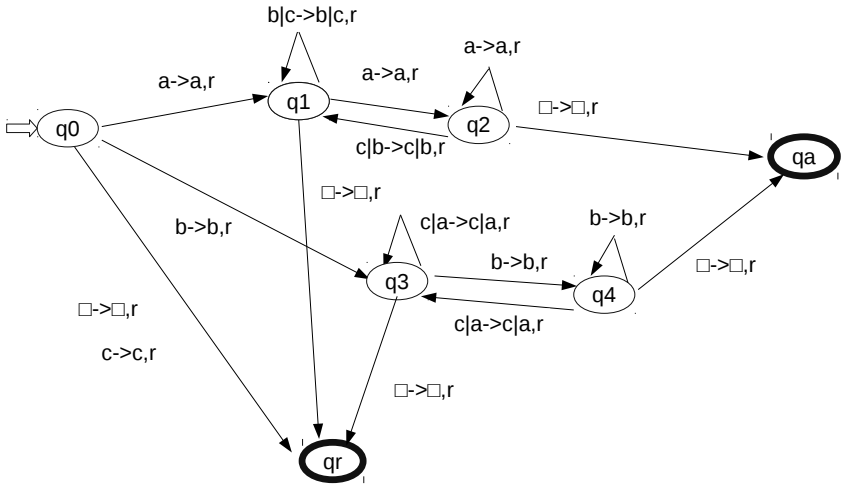

Figure 11: MT qui décide L3

à la clause c_i' qui représente les m-2 derniers littéraux de c_i et nous ajoutons l'équivalence $r \leftrightarrow c_i'$, cette équivalence est traitée recursivement avec les règles r1,r2 et r3.

.3.7 Exercice 7

Pour montrer que le langage $E_{TM} = \{< M > | L(M) = \emptyset\}$ n'est pas décidable, nous créons une réduction à partir de A_{TM}. Supposons que la MT DE est le décideur de E_{TM}, et construisons un décideur DA pour A_{TM} (voir l'algorithme 8):

Il est clair que L(M') est soit $\emptyset$, si M n'accepte pas w ou bien $\{w\}$ si M accepte w, donc le décideur de E_{TM} peut trancher si L(M') est vide ou non et par conséquent peut nous dire si M accepte w ou non.

.3.8 Exercice 8

Pour montrer que $EQ_{TM} = \{< M1, M2 > | L(M1) = L(M2)\}$ n'est pas décidable, nous creons une réduction à partir de E_{TM}. Supposons que la MT DEQ est le décideur de EQ_{TM}, et construisons un décideur DE pour E_{TM} (voir l'algorithme 9):
Notons que le langage de M' est $\emptyset$ et si DEQ arrive à établir l'équivalence entre

Algorithme 8 : X-FNC-TO-3-FNC

Input : $C = \{c_1, c_2, ...c_m\}$: l'ensemble des clauses

C'= C;
C"= $\emptyset$;
$While(C' \neq \emptyset)$ Begin ;
  $c_i = GetClauseFrom(C')$;
  if $(size(c_i) = 1)$ then ;
  add(C", $c_i \vee x1 \vee x2$);add(C", $c_i \vee \neg x1 \vee x2$);
  add(C", $c_i \vee x1 \vee \neg x2$);add(C", $c_i \vee \neg x1 \vee \neg x2$);
  remove(C',c_i);
  endIf ;
  if $(size(c_i) = 2)$ then ;
  add(C", $c_i \vee x1$);add(C", $c_i \vee \neg x1$);
  remove(C',c_i);
  endIf ;
  if $(size(c_i) = 3)$ then ;
  add(C", c_i);
  remove(C',c_i);
  endIf ;
  if $(size(c_i) > 3)$ then ;
  $x_1 = GetFirstLitteral(c_i)$;
$x_2 = GetSecondLitteral(c_i)$;$c'_i = GetRemainingLitterals(c_i)$;
  add(C', $x_1 \vee x_2 \vee c'_i$);
  add(C', $\neg r \vee c'_i$);
  For each litteral $y_j \in c'_i$add(C', $\neg y_k \vee r$);
   remove(C',c_i);
  endIf ;
End;
Return (C");

Algorithme 9 : DA

Input : $(< M >, w)$

create M'(w');
  if (w'≠w) then reject;
  else;
  simulate M on w ;
   If M accepts, then accept;
   If M rejects, then reject;
simulate DE on $< M' >$;
  If DE accept (le langage de M' est vide), then reject;
  If DE reject (le langage de M' n'est pas vide), then accept;

Algorithme 10 : DE

Input : $< M >$

```
1 create M'(w') // M' rejette toutes les chaines ,i.e., L(M')=∅;
2    reject;
3 simulate DEQ on (< M >, < M' >) ;
4    If DEQ accept (le langage de M est vide), then accept;
5    If DEQ reject (le langage de M n'est pas vide), then reject;
```

$\emptyset$ et L(M), alors on peut décider le problème E_{TM}. Mais on sait que E_{TM} est indécidable, donc DEQ ne peut exister.

Références

1. BIERE. A, HEULE. M and MAAREN. H. V, eds. Handbook of satisfiability. IOS press, Vol. 185, 2009.

2. CHURCH. A. A note on the Entscheidungsproblem. Journal of Symbolic Logic. 1:40-41,1936.

3. CHURCH. A. An unsolvable problem of elementary number theory. American Journal of Mathematics, 58:345-363, 1938.

4. COOK. S. A. The complexity of theorem-proving procedures. Proceedings of the third annual ACM symposium on Theory of computing. ACM, 1971.

5. DAVIS. M., LOGEMANN G., LOVELAND D. A Machine Program for Theorem Proving. Journal of the Association for Computing Machinery, vol. 5, p. 394–397, 1962.

6. DE MOURA. L and BJøRNER. N. Z3: An efficient SMT solver. International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2008.

7. EÉN. N. and SÖRENSSON. N. An Extensible SAT-solver. GIUNCHIGLIA E., TACCHELLA A., Eds., SAT, vol. 2919 de Lecture Notes in Computer Science, Springer, p. 502–518, 2003.

8. FLEURENT. C. and FERLAND.J. A. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. Second DIMACS Implementation Challenge : Cliques, Coloring and Satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, p. 619-652, 1996.

9. GENTZEN. G, Untersuchungen über das logische Schließen. I, Mathematische Zeitschrift, vol. 39, 1935, p. 176-210.

10. GOLDBERG. A. On the Complexity of the Satisfiability Problem, Rapport n 16, New York University, 1979

11. GOLDBERG. E and Novikov. Y. BerkMin: a fast and robust SAT-solver. In Design, Automation and Testing in Europe Conference, pages 142–149, March 2002.

12. GOMES. C. P., SELMAN. B and KAUTZ. H. Boosting Combinatorial Search Through Randomization. Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98), Madison, Wisconsin, p. 431–437, 1998.

13. GONZALEZ, T.F. Handbook of approximation algorithms and metaheuristics. Chapman and Hall/CRC, 2007.

14. HAMADI. Y, JABBOUR. S and SAIS. L. ManySAT: a Parallel SAT Solver. JSAT 6.4. 245-262, 2009.

15. HOOS. H.H., STÜTZLE. T. Stochastic Local Search. Foundations and Applications, Morgan Kaufmann, 2005.

16. KIRKPATRICK. S, GELATT. C.D, and VECCHI. M.P. Optimization by simulated annealing. science. Vol 220. No. 4598. 671-680, 1983.

17. KLEENE. S.C. General recursive functions of natural numbers. Mathematische Annalen, 112:727-742,1936.

18. LARDEUX. F., SAUBION. F and HAO. J.-K. GASAT : A Genetic Local Search Algorithm for the Satisfiability Problem. Evolutionary Computation, vol. 14, n 2, p. 223-253, 2006.

19. MARQUES-SILVA. J.P. and SAKALLAH. K.A. GRASP – A New Search Algorithm for Satisfiability, IEEE/ACM International Conference on Computer-Aided Design, 1996.

20. MARQUES-SILVA. J.P. and SAKALLAH. K.A. GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48.5 : 506-521, 1999.

21. MCALLESTER. D., SELMAN. B and KAUTZ. H. Evidence for invariants in local search. Proc. of AAAI-97, p. 321-326, 1997.

22. MOSKEWICZ. M., MADIGAN C., ZHAO. Y., ZHANG. L and MALIK. S. Chaff: Engineering an efficient SAT solver. Proceedings of the 38th annual Design Automation Conference. ACM, 2001.

23. MUNAWAR. A, WAHIB. M, MUNETOMO. M, AKAMA.K. (2009). Hybrid of genetic algorithm and local search to solve MAX-SAT problem using nVidia CUDA framework. Genetic Programming and Evolvable Machines, 10(4), 391

24. RAUTENBERG. W. A concise introduction to mathematical logic. New York, NY: Springer, 2006.

25. ROBINSON. J.A. A Machine-Oriented Logic Based on the Resolution Principle, J. Assoc. Comput. Mach. 12, 23-41, 1965.

26. RYAN. L. Efficient algorithms for clause-learning SAT solvers. Diss. Theses (School of Computing Science)/Simon Fraser University, 2004.

27. SAÏS. L. Problème SAT: progrès et défis. Hermès, 2008.

28. RUSSELL. S.J and NORVIG. P. Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited, 2016.

29. SCHIEX. T and ALLIOT. J.M. Intelligence Artificielle et Informatique théorique. Cépaduès-éd, 1993.

30. TURING. A.M. On computable numbers with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, 42(2):230-265, 1936.

31. VON-NEUMANN. J and BURKS. A.W. Theory of self-reproducing automata. IEEE Transactions on Neural Networks, vol. 5, no 1, p. 3-14, 1966.

32. ZHANG. L., MADIGAN. C., MOSKEWICZ. M. and MALIK. S. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. Proc. of ICCAD, San Jose, 2001.