Cours Logique et Calculabilité

L3 Informatique 2013/2014

Texte par Séverine Fratani, avec addenda par Luigi Santocanale
Version du 4 février 2014

Table des matières

1 Introduction **5**
1.1 Pourquoi la logique? . . . 5
1.1.1 La formalisation du langage . . . 5
1.1.2 La formalisation du raisonnement . . . 5
1.2 Logique et informatique . . . 6
1.3 Contenu du cours . . . 6

2 Calcul propositionnel **7**
2.1 Introduction . . . 7
2.2 Syntaxe du calcul propositionnel : les formules . . . 7
2.3 Sémantique du calcul propositionnel . . . 8
2.3.1 Modèles d'une formule . . . 10
2.3.2 La conséquence logique (d'un ensemble de formules) . . . 12
2.3.3 Décidabilité du calcul propositionnel . . . 17
2.4 Equivalence entre formules . . . 18
2.4.1 La substitution . . . 18
2.4.2 Equivalences classiques . . . 18
2.4.3 Formes normales . . . 19
2.5 Le problème SAT . . . 21
2.5.1 Définition du problème . . . 21
2.5.2 Un problème NP-complet . . . 21
2.5.3 Modélisation - Réduction à SAT . . . 23
2.5.4 Algorithmes de résolution de SAT . . . 25
2.5.5 Sous-classes de SAT . . . 30
2.5.6 Les SAT-solvers . . . 32
2.5.7 Applications . . . 33
2.5.8 Sur la modélisation . . . 35
2.6 Systèmes de preuves . . . 36
2.6.1 La notion de système formel . . . 36
2.6.2 La méthode de la coupure . . . 36
2.6.3 Systèmes de preuve à la Hilbert . . . 40
2.7 Résumé . . . 45

3 Calcul des prédicats **49**
3.1 Introduction . . . 49
3.2 Préliminaires . . . 49
3.2.1 Les fonctions . . . 49
3.2.2 Les relations . . . 50
3.3 Un exemple . . . 50
3.3.1 Interprétation 1 . . . 50
3.3.2 Interprétation 2 . . . 51
3.3.3 Interprétation 3 . . . 51
3.3.4 Interprétation 4 . . . 51
3.3.5 Interprétation 5 . . . 51

3.3.6 Interprétation 6 . . . 52
3.3.7 Comparaison des interprétations . . . 52
3.4 Expressions et formules . . . 52
3.4.1 Les termes . . . 52
3.4.2 Le langage . . . 53
3.4.3 Les formules du calcul des prédicats . . . 54
3.4.4 Occurrences libres et liées d'une variable . . . 55
3.5 Sémantique . . . 56
3.5.1 Structures et valuations . . . 56
3.5.2 Evaluation . . . 57
3.5.3 Un exemple . . . 58
3.5.4 Vocabulaire . . . 59
3.6 Manipulation de formules . . . 60
3.6.1 Substitution de variables . . . 60
3.6.2 Equivalences classiques . . . 60
3.6.3 Formes Normales . . . 61
3.7 Unification . . . 63
3.7.1 Substitions et MGUs . . . 63
3.7.2 Algorithme d'unification . . . 64
3.7.3 Correction et completude . . . 65
3.8 Résolution . . . 67
3.8.1 Substitution, sur les formules propostionnelles . . . 67
3.8.2 Les règles du calcul de la résolution . . . 68
3.8.3 Utilisation d'un démonstrateur automatique . . . 71

4 Calculabilité 77
4.1 Machines de Turing . . . 77
4.2 Problèmes de décision . . . 78
4.3 Un problème indécidable . . . 79
4.4 Thèse de Chruch . . . 79
4.5 Indecidabilité de la logique du premier ordre . . . 79

Chapitre 1

Introduction

1.1 Pourquoi la logique ?

1.1.1 La formalisation du langage

Le mot *logique* provient du grec *logos* (raison, discours), et signifie "science de la raison". Cette science a pour objets d'étude le discours et le raisonnement. Ceux-ci dépendent bien entendu du langage utilisé. Si on prend le langage courant, on se rend compte facilement :

- qu'il contient de nombreuses ambiguïtés : nous ne sommes pas toujours sûrs de la sémantique d'un énoncé ou d'une phrase. Par exemple : « nous avons des jumelles à la maison ». (Deux filles ou des lunettes optiques ?) Ou « il a trouvé un avocat » (le professionnel ou le fruit ?)
- qu'il est difficile de connaître la véracité d'un énoncé : « il pleuvra demain », « Jean est laid », « la logique c'est dur ».
- qu'il permet d'énoncer des choses paradoxales :
  1. « Je mens » : comme pour tout paradoxe de ce type, on aboutit à la conclusion que si c'est vrai alors c'est faux... et inversement.
  2. « Je suis certain qu'il n'y a rien de certain »
  3. Un arrêté enjoint au barbier (masculin) d'un village de raser tous les hommes du village qui ne se rasent pas eux-mêmes et seulement ceux-ci. Le barbier n'a pas pu respecter cette règle car :
     - s'il se rase lui-même, il enfreint la règle, car le barbier ne peut raser que les hommes qui ne se rasent pas eux-mêmes ;
     - s'il ne se rase pas lui-même (qu'il se fasse raser ou qu'il conserve la barbe), il est en tort également, car il a la charge de raser les hommes qui ne se rasent pas eux-mêmes.
  4. Paradoxe de Russel : c'est la version mathématique du paradoxe du barbier : soit a l'ensemble des ensembles qui ne se contiennent pas eux-même. Cet ensemble n'existe pas car on peut vérifier que $a \in a$ ssi $a \notin a$.

Tout ceci fait que les langues naturelles ne sont pas adaptées au raisonnement formel. C'est pourquoi par exemple, on vous a appris un langage spécifique pour faire des preuves en mathématique. Une preuve mathématique ne peut être faite en utilisant tout le vocabulaire de la langue naturelle car les énoncés et les preuves deviendraient alors ambiguës. Le langage utilisé en mathématique est celui de la logique classique (en réalité, c'est un langage un peu plus souple, entre la logique classique et la langue naturelle).

1.1.2 La formalisation du raisonnement

Une fois le langage formalisé, ce qui intéresse les logiciens c'est le raisonnement, et en particulier, la définition de systèmes formels permettant de mécaniser le raisonnement. Au début du 20ième siècle, le rêve du logicien est de faire de la logique un calcul et de mécaniser le raisonnement et par suite toutes les mathématiques. En 1930, Kurt Gödel met fin à cette utopie en présentant son résultat d'incomplétude : il existe des énoncés d'arithmétique qui ne sont pas prouvable par un

système formel de preuve. Il n'existe donc pas d'algorithme qui permette de savoir si un énoncé mathématique est vrai.

1.2 Logique et informatique

Malgré ce résultat négatif, l'arrivée de l'informatique à partir des années 30 marque l'essor de la logique.

Elle est présente dans quasiment tous les domaines de l'informatique :

- vous verrez par exemple en cours d'architecture que votre ordinateur est formé de circuits logiques.
- la programmation n'est au fond que de la logique. Dans les année 60, la correspondance de Curry-Howard, établie une correspondance preuve/programme : une relation entre les démonstrations formelles d'un système logique et les programmes d'un modèle de calcul.
- le traitement automatique des langues,
- l'intelligence artificielle,
- la logique apparait également dans toutes les questions de sureté.
  On demande maintenant de plus en plus de prouver la sureté des programmes et des protocoles. Pour cela on modélise les exécutions des programmes, on exprime les propriétés de sûreté par une formule logique, puis on vérifie que les modèles satisfont bien la formule.
  A cette effet, d'innombrables logiques ont étés développées, comme les logiques temporelles qui permettent de raisonner sur l'évolution de certains systèmes au cours du temps. Il existe même des logiques pour formaliser les règles des pare-feu, afin d'éviter d'avoir des systèmes de règle incohérents.
- et plein d'autres que j'oublie.

Il existe également des logiciels qui permettent de prouver des formules logique (automatiquement ou semi-automatiquement). En particulier on a des logiciels permettant de générer du code vérifié : on entre une abstraction du programme a réaliser, on prouve sur cette abstraction de façon plus ou moins automatique mais sûre, les propriétés de sûreté souhaitées ; et le logiciel produit du code certifié.

L'informatique est donc indissociable de la logique. Heureusement tout bon informaticien n'est pas obligé d'être un bon théoricien de la logique, mais il doit être capable maitriser son utilisation.

1.3 Contenu du cours

On s'intéressera principalement à (des fragments de) la logique classique, qui est la logique utilisée pour les mathématiques, et forme la base de presque toutes les autres logiques.

Nous allons nous intéresser aux fragment suivants :

- la logique propositionnelle ;
- la logique du premier ordre.

Pour chacune de ces logiques, nous nous poserons principalement les questions suivantes :

- Quelle est sa syntaxe ? I.e., comment écrire une phrase dans le langage de la logique considéré.
- Quel est sa sémantique ? C'est-à-dire, étant une phrase, savoir lui attribuer un sens. Cette question ouvre à une autre qui est "de quel forme sont les modèles d'une formule", c'est à dire : mon langage parle d'objets, qui se placent dans un univers précis : quel est cet univers ?
- La logique est elle-décidable ? Etant donné une phrase (formule) du langage, existe-il une procédure effective permettant d'évaluer cette formule.
- Existe-il un système formel de calcul permettant de "prouver" qu'un énoncé est vrai ou faux.

D'autres questions se posent évidement mais se sont principalement celles-ci qui intéressent l'informaticien.

Chapitre 2

Calcul propositionnel

2.1 Introduction

Les formules (ou phrases, ou énoncés) du calcul propositionnel sont de deux types : ou bien une formule est une *proposition atomique*, ou bien elle est composée à partir d'autres formules à l'aide des connecteurs logiques $\wedge, \neg, \vee, \Rightarrow$ (*et*, *non*, *ou*, *implique*, que l'on appelle *connecteurs propositionnels*).

Considérons, par exemple, l'énoncé arithmétique « 2+2 = 4 *ou* 3+3 = 5 ». Cet énoncé peut se considérer comme construit des propositions atomiques « 2+2 = 4 » et « 3+3 = 5 », via le connecteur propositionnel *ou*. Une analyse similaire peut se faire pour les énoncés du langage naturel. On considère l'énoncé « s'il pleut, alors le soleil se cache », que l'on reconnaîtra être équivalent à « il pleut implique que le soleil se cache », comme obtenu des deux propositions atomiques « s'il pleut » et « le soleil se cache » via le connecteur propositionnel *implique*.

Une proposition atomique est un énoncé simple, ne pouvant prendre que les valeurs "vrai" ou "faux ", et ce de façon non ambiguë ; elle donne donc une information sur un état de chose. De plus une proposition atomique est indécomposable : « le ciel est bleu et l'herbe est verte » n'est pas une proposition atomique mais la composition de deux propositions atomiques. Dans l'analyse du langage naturel, on ne peut pas considérer comme des propositions : les souhaits, les phrases impératives ou les interrogations.

Nous avons déjà vu des exemples de formules composées. Considérons maintenant l'énoncé « s'il neige, alors le soleil se cache et il fait froid ». C'est une formule composée, via le connecteur *implique*, depuis la formule atomique « il neige » et la formule composée « le soleil se cache et il fait froid ». On peut donc composer des formules à partir d'autres formules composées. La valeur de vérité d'une formule composée se calcule comme une fonction des formules dont elle est composée.

Le calcul des propositions est la première étape dans la définition de la logique et du raisonnement. Il définit les règles de déduction qui relient les phrases entre elles, sans en examiner le contenu ; il est ainsi une première étape dans la construction du calcul des prédicats, qui lui s'intéresse au contenu des propositions.

Nous partirons donc en général de faits : "p est vrai, q est faux" et essaierons de déterminer si une affirmation particulière est vraie.

2.2 Syntaxe du calcul propositionnel : les formules

Le *langage* du calcul propositionnel est formé de :
— symboles propositionnels $\textsc{Prop} = \{p_1, p_2, \ldots\}$;
— connecteurs logiques $\{\neg, \wedge, \vee, \Rightarrow\}$;
— symboles auxiliaires : parenthèses et espace.

Remarque 2.1. Dans la littérature logique on utilise plusieurs synonymes pour symbole propositionnel ; ainsi *variable propositionnelle*, *proposition atomique*, *formule atomique*, ou encore *atome* sont tous des synonymes de *symbole propositionnel.*

L'ensemble $\mathcal{F}_{\text{cp}}$ des *formules* du calcul propositionnel est le plus petit ensemble tel que :
— tout symbole propositionnel est une formule ;
— si φ est une formule alors $\neg\varphi$ est une formule ;
— si φ, ψ sont des formules alors $\varphi \vee \psi$, $\varphi \wedge \psi$ et $\varphi \Rightarrow \psi$ sont des formules.

Les symboles auxiliaires ne sont utilisés que pour lever les ambiguïtés possibles : par exemple, la formule $p \vee q \wedge r$ est ambiguë, car elle peut se lire de deux façons différentes, $((p \vee q) \wedge r)$ ou bien $(p \vee (q \wedge r))$.

Exemple 2.2. p, $p \Rightarrow (q \vee r)$ et $p \vee q$ sont des formules propositionnelles ; $\neg(\vee q)$ et $f(x) \Rightarrow g(x)$ n'en sont pas.

A cause de la structure inductive de la définition, une formule peut-être vue comme un arbre dont les feuilles sont étiquetés par des symboles propositionnels et les noeuds par des connecteurs. Par exemple, la formule $p \Rightarrow (\neg q \wedge r)$ correspond à l'arbre représenté Figure 2.2.

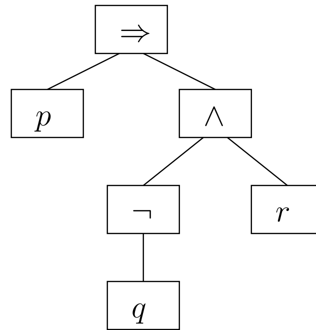


FIGURE 2.1 – Représentation arborescente de la formule $p \Rightarrow (\neg q \wedge r)$

Notation 2.3. On utilise souvent en plus le connecteur binaire $\Leftrightarrow$ comme abréviation : $\varphi \Leftrightarrow \psi$ est l'abréviation de $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$. De la même façon, on ajoute le symbole $\bot$ qui correspond à *Faux* et le symbole $\top$ qui correspond à *Vrai*. Ces deux symboles sont aussi des abréviations, ils ne sont pas indispensables au langage. (Par exemple $\bot$ peut être utilisé à la place de $p \wedge \neg p$ et $\top$ à la place de $p \vee \neg p$.)

Définition 2.4 (Sous-formule)**.** L'ensemble $SF(\varphi)$ des sous-formules d'une formules φ est défini par induction de la façon suivante.
— $SF(p) = \{p\}$;
— $SF(\neg\varphi) = \{\neg\varphi\} \cup SF(\varphi)$;
— $SF(\varphi \circ \psi) = \{\varphi \circ \psi\} \cup SF(\varphi) \cup SF(\psi)$ (où $\circ$ désigne un des symboles $\wedge$, $\vee$, $\Rightarrow$).

Par exemple, $SF(p \Rightarrow (\neg q \wedge r)) = \{p, q, r, \neg q, \neg q \wedge r, p \Rightarrow (\neg q \wedge r)\}$. Quand on voit une formule comme un arbre, une sous-formule est simplement un sous-arbre (voir Figure 2.2).

Définition 2.5 (Sous-formule stricte)**.** ψ est une sous-formule stricte de φ si ψ est une sous-formule de φ qui n'est pas φ.

2.3 Sémantique du calcul propositionnel

Il faut maintenant un moyen de déterminer si une formule est vraie ou fausse. La première étape est de donner une valeur de vérité aux propositions atomiques. L'évaluation d'une formule, dépend donc des valeurs choisies pour les symboles propositionnels. Ces valeurs sont données par une **valuation**.

Définition 2.6 (Valuation)**.** Une valuation est une application de PROP dans $\{0, 1\}$. La valeur 0 désigne le "faux" et la valeur 1 désigne le "vrai".

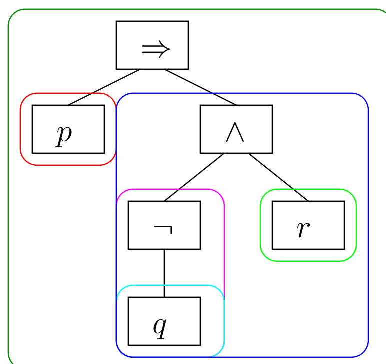


FIGURE 2.2 – Représentation arborescente des sous-formules de $p \Rightarrow (\neg q \wedge r)$

Une valuation sera souvent donnée sous forme d'un tableau. Par exemple, si PROP $= \{p, q\}$ alors la valuation $v : p \mapsto 1,\ q \mapsto 0$ s'écrit plus simplement v :

p	q
1	0

Une fois la valuation v choisie, la valeur de la formule se détermine de façon naturelle, par extension de la valuation v aux formules de la façon suivante :

Définition 2.7 (Valeur d'une formule)**.**
— $v(\neg\varphi) = 1$ ssi $v(\varphi) = 0$;
— $v(\varphi \vee \psi) = 1$ ssi $v(\varphi) = 1$ ou $v(\psi) = 1$;
— $v(\varphi \wedge \psi) = 1$ ssi $v(\varphi) = 1$ et $v(\psi) = 1$;
— $v(\varphi \Rightarrow \psi) = 0$ ssi $v(\varphi) = 1$ et $v(\psi) = 0$.

La définition précédente peut apparaître trompeuse car circualire : afin d'expliquer la logique, nous somme en train de l'utiliser (*ssi*, *ou*, *et* ...). Par ailleurs, on peut se servir de la définition suivante qui est en effet équivalente à la Définition 2.7 :

Définition 2.8 (Valeur d'une formule (bis))**.**
— $v(\neg\varphi) = 1 - v(\varphi)$;
— $v(\varphi \vee \psi) = \max(v(\varphi), v(\psi))$;
— $v(\varphi \wedge \psi) = \min(v(\varphi), v(\psi))$;
— $v(\varphi \Rightarrow \psi) = v(\neg\varphi \vee \psi)$.

Cette dernière définition est purement combinatoire car elle repose sur la structure de l'ensemble ordonné *fini* $\{ 0 < 1 \}$; nous supposons en fait que cette structure est évidente et claire par soi-même qu'il n'y a pas besoin de la justifier par d'autres moyens.

Exercice 2.9. Proposez un algorithme qui, étant donné une formule φ du calcul propositionnel et une valuation v, calcule $v(\varphi)$. Quel type de structure de données utiliser pour coder les formules ? Quel type de structure de données utiliser pour coder les valuations ?

Notez que la définition 2.8 correspond aux tables de vérité des connecteurs logiques (dont vous avez sûrement entendu parler) :

p	$\neg p$
0	1
1	0

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

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

p	q	$p \Rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Remarque 2.10 (Langage naturel et langage formel)**.** Remarquez la définition particulière de l'implication : on l'entend en général comme un "si ..., alors ...", on voit ici que l'énoncé "si 1+1=1, alors la capitale de la France est Marseille" est vrai, puisque toute phrase $\varphi \Rightarrow \psi$ est vraie dès

lors que φ est évaluée à faux. Ceci est peu naturel, car dans le langage courant, on ne s'intéresse à la vérité d'un tel énoncé que lorsque la condition est vraie : "s'il fait beau je vais à la pêche" n'a d'intérêt pratique que s'il fait beau... Attribuer la valeur vrai dans le cas ou la prémisse est fausse correspond a peu près à l'usage du si .. alors dans la phrase suivante : "Si Pierre obtient sa Licence, alors je suis Einstein" : c'est à dire que partant d'une hypothèse fausse, alors je peux démontrer des choses fausses (ou vraies). Par contre, il n'est pas possible de démontrer quelque chose de faux partant d'une hypothèse vraie.

D'autres exemples où il est difficile de coder le langage naturel via le langage formel :

— comment coderiez vous, en langage formel, l'énoncé français « Soit il est frais, soit il est chaud » ?

— et comment coderiez vous l'énoncé anglais « Either I cannot understand French, or my professor doesn't know how to speak it » ?

On peut ajouter la définition de la valeur de l'abréviation $\Leftrightarrow$: $v(\varphi \Leftrightarrow \psi) = 1$ ssi $v(\varphi) = v(\psi)$. Ce qui correspond à la table de vérité suivante :

p	q	$p \Leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

Exercice 2.11. Définissons l'ensemble $\textsc{Prop}(\varphi)$, des variables propositionnelles contenues dans $\varphi \in \mathcal{F}_{\text{cp}}$, par induction comme suit :

$$\begin{aligned} \textsc{Prop}(p) &= \{\, p \,\}\,, \\ \textsc{Prop}(\neg\varphi) &= \textsc{Prop}(\varphi)\,, \\ \textsc{Prop}(\varphi \circ \psi) &= \textsc{Prop}(\varphi) \cup \textsc{Prop}(\psi)\,, & \circ \in \{\, \wedge, \vee, \Rightarrow \,\}\,. \end{aligned}$$

Montrez que :

— $\textsc{Prop}(\varphi) = \textsc{Prop} \cap SF(\varphi)$, pour tout $\varphi \in \mathcal{F}_{\text{cp}}$;

— si $v(p) = v'(p)$ pour tout $p \in \textsc{Prop}(\varphi)$, alors $v(\varphi) = v'(\varphi)$.

2.3.1 Modèles d'une formule

Définition 2.12. L'ensemble des **valuations** d'un ensemble de variables propositionnelles $\textsc{Prop}$ est noté $\mathsf{Val}(\textsc{Prop})$ (ou juste Val lorsqu'il n'y a pas d'ambiguité sur $\textsc{Prop}$). $\mathsf{Val}(\textsc{Prop})$ est donc l'ensemble des fonctions de $\textsc{Prop}$ dans $\{0, 1\}$.

Par exemple, si $\textsc{Prop} = \{p, q, r\}$, alors Val est représenté par la Table 2.3.1, dans lequel chaque ligne est une valuation de $\textsc{Prop}$:

p	q	r
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

TABLE 2.1 – L'ensemble $\mathsf{Val}(\textsc{Prop})$ des valuations de $\textsc{Prop} = \{p, q, r\}$

Définition 2.13 (Modèle d'une formule)**.** Un **modèle** de φ est une valuation v telle que $v(\varphi) = 1$. On note $\mathsf{mod}(\varphi)$ l'ensemble des modèles de φ.

Exemple 2.14. Si PROP $= \{p, q, r\}$ et $\varphi = (p \vee q) \wedge (p \vee \neg r)$ alors l'ensemble des modèles de φ est

$\mathsf{mod}(\varphi) =$

p	q	r
0	1	0
1	0	0
1	1	1
1	1	0
1	1	1

Définition 2.15 (Satisfaisabilité)**.** Une formule φ est **satisfaisable** (ou **consistante**, ou encore **cohérente**) si elle admet un modèle (*i.e.*, s'il existe une valuation v telle que $v(\varphi) = 1$, *i.e.* si $\mathsf{mod}(\varphi) \neq \varnothing$).

Définition 2.16 (Insatisfaisabilité)**.** Une formule φ est **insatisfaisable** (ou **inconsistante**, ou **incohérente**) si elle n'admet aucun modèle (*i.e.*, si pour toute valuation v, $v(\varphi) = 0$, *i.e.*, si $\mathsf{mod}(\varphi) = \varnothing$)

Définition 2.17 (Tautologie)**.** Une formule φ est une **tautologie** (ou **valide**) si $v(\varphi) = 1$ pour toute valuation v (*i.e.*, si $\mathsf{mod}(\varphi) = \mathsf{Val}$). On note $\models \varphi$ pour dire que φ est une tautologie.

Un exemple de tautologie est $\varphi \vee \neg\varphi$, c'est à dire le *tiers exclus.*

Exercice 2.18. Montrez que les formules suivantes sont des tautologies :

$$p \Rightarrow p\,, \quad p \Rightarrow (q \Rightarrow p)\,, \quad (p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))\,, \quad ((p \Rightarrow q) \Rightarrow p) \Rightarrow p\,.$$

Définition 2.19 (Equivalence)**.** On dit que φ **est équivalente à** ψ si les deux formules ont les mêmes modèles (*i.e.* si $\mathsf{mod}(\varphi) = \mathsf{mod}(\psi)$). On note alors $\varphi \equiv \psi$.

Exemple 2.20. Les opérateurs $\wedge$, $\vee$ sont associatifs-commutatifs. Deux formules identiques à associativité-commutativité près sont équivalentes. Remplacer une sous-formule ψ d'une formule φ par une formule équivalente ψ' donne une formule notée $\varphi[\psi \leftarrow \psi']$. Cette substitution préserve les modèles, *i.e.*, $\mathsf{mod}(\varphi) = \mathsf{mod}(\varphi[\psi \leftarrow \psi'])$.

Exercice 2.21. Prouvez les équivalences suivantes :

$$\begin{array}{ccc}
\varphi \vee \bot \equiv \varphi & \varphi \wedge \bot \equiv \bot & (\varphi \wedge \psi) \wedge \theta \equiv \varphi \wedge (\psi \wedge \theta) \\
\varphi \vee \top \equiv \top & \varphi \wedge \top \equiv \varphi & (\varphi \wedge \psi) \vee \theta \equiv (\varphi \wedge \theta) \vee (\psi \wedge \theta) \\
\varphi \vee \psi \equiv \psi \vee \varphi & \varphi \wedge \psi \equiv \psi \wedge \varphi & (\varphi \vee \psi) \wedge \theta \equiv (\varphi \wedge \theta) \vee (\varphi \wedge \theta) \\
\varphi \vee \varphi \equiv \varphi & \neg(\varphi \vee \psi) \equiv (\neg\varphi) \wedge (\neg\psi) & (\varphi \vee \psi) \vee \theta \equiv \varphi \vee (\psi \vee \theta) \\
\neg\neg\varphi \equiv \varphi & \varphi \wedge \varphi \equiv \varphi & \neg(\varphi \wedge \psi) \equiv (\neg\varphi) \vee (\neg\psi)\,.
\end{array}$$

Proposition 2.22. *Soient φ et ψ deux formules, on a :*

1. $\mathsf{mod}(\neg\varphi) = \mathsf{Val} - \mathsf{mod}(\varphi)$ *;*
2. $\mathsf{mod}(\varphi \vee \psi) = \mathsf{mod}(\psi) \cup \mathsf{mod}(\psi)$ *;*
3. $\mathsf{mod}(\varphi \wedge \psi) = \mathsf{mod}(\psi) \cap \mathsf{mod}(\psi)$ *;*
4. $\models \varphi \Rightarrow \psi$ *ssi* $\mathsf{mod}(\varphi) \subseteq \mathsf{mod}(\psi)$.

Démonstration. 1. pour toute valuation $v \in \mathsf{Val}$,

$$\begin{array}{rll}
v \in \mathsf{mod}(\neg\varphi) & \text{ssi} & v(\neg\varphi) = 1 \\
& \text{ssi} & v(\varphi) = 0 \\
& \text{ssi} & v \notin \mathsf{mod}(\varphi) \\
& \text{ssi} & \mathsf{Val} - \mathsf{mod}(\varphi)
\end{array}$$

2. pour toute valuation $v \in \mathsf{Val}$,

$$\begin{array}{rll}
v \in \mathsf{mod}(\varphi \vee \psi) & \text{ssi} & v(\varphi) = 1 \text{ ou } v(\psi) = 1 \\
& \text{ssi} & v \in \mathsf{mod}(\varphi) \text{ ou } v \in \mathsf{mod}(\psi) \\
& \text{ssi} & \mathsf{mod}(\varphi) \cup \mathsf{mod}(\psi)
\end{array}$$

3. pour toute valuation $v \in \mathsf{Val}$,

$$\begin{aligned} v \in \mathsf{mod}(\varphi \wedge \psi) \quad & \text{ssi} \quad v(\varphi) = 1 \text{ et } v(\psi) = 1 \\ & \text{ssi} \quad v \in \mathsf{mod}(\varphi) \text{ et } v \in \mathsf{mod}(\psi) \\ & \text{ssi} \quad \mathsf{mod}(\varphi) \cap \mathsf{mod}(\psi) \end{aligned}$$

4.

$$\begin{aligned} \models \varphi \Rightarrow \psi \quad & \text{ssi} \quad \text{pour toute valuation } v \in \mathsf{Val}, v(\varphi \Rightarrow \psi) = 1 \\ & \text{ssi} \quad \text{pour toute valuation } v \in \mathsf{Val}, v(\neg\varphi \vee \psi) = 1 \\ & \text{ssi} \quad \text{pour toute valuation } v \in \mathsf{Val}, v(\varphi) = 0 \text{ ou } v(\psi) = 1 \\ & \text{ssi} \quad \text{pour toute valuation } v \in \mathsf{Val}, v(\varphi) \leq v(\psi) \\ & \text{ssi} \quad \mathsf{mod}(\varphi) \subseteq \mathsf{mod}(\psi) \quad \square \end{aligned}$$

Définition 2.23 (Conséquence logique)**.** Une formule ψ est **conséquence logique** d'une formule φ si tout modèle de φ est un modèle de ψ (*i.e.*, si $\mathsf{mod}(\varphi) \subseteq \mathsf{mod}(\psi)$). On note alors $\varphi \models \psi$.

Remarque 2.24. Attention à la confusion dans les deux notations !

— $v \models \varphi$ où v est une valuation, i.e., l'assignation d'une valeur aux propositions atomiques de la formule ; c'est un raccourcis assez fréquent pour $v(\varphi) = 1$;

— $\psi \models \varphi$ où ψ est une formule.

Proposition 2.25. *Soient φ et ψ deux formules propositionnelles.*

1. $\varphi \models \psi$ si et seulement si $\models \varphi \Rightarrow \psi$.

2. $\varphi \equiv \psi$ si et seulement si $\models \varphi \Leftrightarrow \psi$.

Démonstration. 1. Conséquence directe du point 4 de la Proposition 2.22

2. $\models \varphi \Leftrightarrow \psi$ ssi $\forall v \in \mathsf{Val}$, $v(\varphi \Leftrightarrow \psi) = 1$ (par la définition de tautologie) ssi $\forall v \in \mathsf{Val}$, $v(\varphi) = v(\psi)$ (par la table de verité de $\Leftrightarrow$) ssi $\forall v \in \mathsf{Val}$, $v \in \mathsf{mod}(\varphi)$ ssi $v \in \mathsf{mod}(\psi)$ ssi $\mathsf{mod}(\varphi) = \mathsf{mod}(\psi)$ ssi $\psi \equiv \varphi$. $\square$

2.3.2 La conséquence logique (d'un ensemble de formules)

Les formules propositionnelles peuvent être vues comme des contraintes sur les propositions atomiques. Par exemple, $p \wedge q$ contraint p et q à être vraies, où $p \Rightarrow q$ contraint q à être vraie toute fois que p est vraie. Il est donc très courant de considérer des ensembles de formules propositionnelles pour modéliser des problèmes de satisfaction de contraintes. Une valuation satisfaisant toute formule de l'ensemble pourra donc se considérer comme une solution du problème.

On étend les définitions vues précédemment aux ensembles de formules.

Définition 2.26 (Modèle)**.** Un **modèle** d'un ensemble de formules Γ est une valuation v telle que $v(\varphi) = 1$ pour tout $\varphi \in \Gamma$. On note $\mathsf{mod}(\Gamma)$ l'ensemble des modèles de Γ.

Cet ensemble de modèles est donc l'ensemble des valuations qu'on peut attribuer aux variables si on veut respecter toutes les contraintes de Γ.

Définition 2.27 (Satisfaisabilité/Consistance, Insatisfaisabilité/Contradiction)**.** Un ensemble de formules Γ est

— **satisfaisable** (ou **consistant**, ou **cohérent**) s'il admet au moins un modèle (*i.e.*, si $\mathsf{mod}(\Gamma) \neq \varnothing$) ;

— **insatisfaisable** (ou **contradictoire**, ou **inconsistant**, ou encore **incohérent**) s'il n'admet aucun modèle (*i.e.*, si $\mathsf{mod}(\Gamma) = \varnothing$), on note alors $\Gamma \models \bot$.

Un ensemble Γ contradictoire ne peut être satisfait : par exemple l'ensemble $\Gamma = \{p, \neg p\}$ est insatisfaisable.

Définition 2.28 (Conséquence logique)**.** Une formule φ est conséquence logique de Γ si et seulement si toute valuation qui donne 1 à toutes les formules de Γ donne 1 à φ (*i.e.*, si $\mathsf{mod}(\Gamma) \subseteq \mathsf{mod}(\varphi)$), on note alors $\Gamma \models \varphi$. On note cons(Γ) l'ensemble des conséquences logiques de Γ.

Remarque 2.29. Attention aux deux notations :
- $\psi \models \varphi$ où ψ est une formule ;
- $\Gamma \models \varphi$ où Γ est un ensemble de formule.

Remarquez, par ailleurs que $\varphi \models \psi$ si, et seulement si, $\{ \varphi \} \models \psi$; les deux dernières notations sont donc cohérentes entre elles.

Voici des relations élémentaires entre les relations que nous venons de présenter.

Proposition 2.30. $\Gamma \models \varphi$ *ssi* $\Gamma \cup \{\neg\varphi\}$ *est contradictoire.*

Démonstration. $\Gamma \models \varphi$ ssi
pour toute valuation v
- soit v est un modèle de Γ et $v(\varphi) = 1$
- soit v n'est pas un modèle de Γ

ssi pour toute valuation v
- soit v est un modèle de Γ et $v(\neg\varphi) = 0$
- soit v n'est pas un modèle de Γ

ssi pour toute valuation v, v n'est pas un modèle de $\Gamma \cup \{\neg\varphi\}$
ssi $\Gamma \cup \{\neg\varphi\}$ est contradictoire. □

Proposition 2.31. *Pour tous ensembles de formules* Σ, Γ,

$$\mathit{mod}(\Sigma \cup \Gamma) = \mathit{mod}(\Sigma) \cap \mathit{mod}(\Gamma).$$

En particulier, si $\Sigma \subseteq \Gamma$ *alors* $\mathit{mod}(\Gamma) \subseteq \mathit{mod}(\Sigma)$.

Cette proposition se comprend bien si on voit un ensemble de formules comme un ensemble de contraintes sur les variables propositionnelles. Plus on ajoute de contraintes, et moins il reste de possibilités pour résoudre ces contraintes.

Démonstration de la Proposition 2.31. Pour toute valuation v :

$$\begin{aligned} v \in \mathsf{mod}(\Sigma \cup \Gamma) \text{ ssi } & \text{ pour tout } \varphi \in \Sigma \cup \Gamma, v(\varphi) = 1 \\ \text{ssi } & \text{ pour tout } \varphi \in \Sigma, v(\varphi) = 1 \text{ et pour tout } \psi \in \Gamma,\ v(\psi) = 1 \\ \text{ssi } & v \in \mathsf{mod}(\Sigma) \text{ et } v \in \mathsf{mod}(\Gamma) \\ \text{ssi } & v \in \mathsf{mod}(\Sigma) \cap \mathsf{mod}(\Gamma). \end{aligned}$$

□

La preuve de la Proposition suivante est laissée en exercice.

Proposition 2.32. *Si* $\Gamma' \subseteq \Gamma$ *et* $\Gamma' \models \varphi$, *alors* $\Gamma \models \varphi$.

Proposition 2.33. *Soit* $\Gamma = \{ \varphi_1, \ldots, \varphi_n \}$ *un ensemble fini de formules. Nous avons alors*

$$\mathit{mod}(\Gamma) = \mathit{mod}(\varphi_1 \wedge \ldots \wedge \varphi_n)\,,$$

de façon que $\{\varphi_1, \ldots \varphi_n\} \models \psi$ *ssi* $\varphi_1 \wedge \ldots \wedge \varphi_n \models \psi$, *ssi* $(\varphi_1 \wedge \ldots \wedge \varphi_n) \Rightarrow \psi$ *est une tautologie.*

Cette proposition exprime le fait qu'un ensemble **fini** de formules propositionnelles peut toujours être vu comme une seule formule formée de la conjonction des formules de l'ensemble. Une formule étant un objet fini, ce résultat ne se généralise pas au cas des ensembles de taille non bornée. Dans le cas où Γ est infini, il faudra utiliser le théorème de compacité (Théorème 2.39) qui permet de ramener les problèmes de satisfaisabilité et de contradiction d'un ensemble de taille quelconque à celle d'ensemble finis.

Démonstration. Remarquons que

$$\begin{aligned}\mathsf{mod}(\{\,\varphi_1,\ldots,\varphi_n\,\}) &= \mathsf{mod}(\{\,\varphi_1\,\}\cup\ldots\cup\{\,\varphi_n\,\})\\ &= \mathsf{mod}(\{\,\varphi_1\,\})\cap\ldots\cap\mathsf{mod}(\{\,\varphi_n\,\}) && \text{(par la Proposition 2.31)}\\ &= \mathsf{mod}(\varphi_1)\cap\ldots\cap\mathsf{mod}(\varphi_n) && \text{(par la Remarque 2.29)}\\ &= \mathsf{mod}(\varphi_1\wedge\ldots\wedge\varphi_n)\,. && \text{(par la Propositon 2.22)}\end{aligned}$$

Donc, on a que $\mathsf{mod}(\{\,\varphi_1,\ldots,\varphi_n\,\}) \subseteq \mathsf{mod}(\psi)$ si et seulement si $\mathsf{mod}(\varphi_1\wedge\ldots\wedge\varphi_n) \subseteq \mathsf{mod}(\psi)$ et, par la Proposition 2.22, la dernière relation est vraie ssi $(\varphi_1\wedge\ldots\wedge\varphi_n) \Rightarrow \psi$ est une tautologie. □

Proposition 2.34. *$\Gamma \models \varphi$ si, et seulement si, $\mathsf{mod}(\Gamma) = \mathsf{mod}(\Gamma\cup\{\varphi\})$.*

La proposition peut se comprendre comme suit. Une conséquence logique φ d'un ensemble Γ est une nouvelle contrainte déduite directement de Γ. Puisqu'elle découle de Γ, elle ne peut pas apporter des "vraies" contraintes supplémentaires que celles apportées par Γ. Cela signifie que les modèles de Γ et ceux de $\Gamma\cup\{\varphi\}$ sont exactement les mêmes.

Démonstration de la Proposition 2.34. La proposition découle du fait que $\mathsf{mod}(\Gamma\cup\{\varphi\}) = \mathsf{mod}(\Gamma)\cap \mathsf{mod}(\{\varphi\})$, et que la relation $\mathsf{mod}(\Gamma) \subseteq \mathsf{mod}(\varphi)$ est équivalente à $\mathsf{mod}(\Gamma)\cap\mathsf{mod}(\{\varphi\}) = \mathsf{mod}(\Gamma)$. □

Exemple 2.35. L'ensemble $\Gamma = \{(p \Rightarrow s) \vee q, \neg q\}$ possède comme conséquence logique $p \Rightarrow s$. Bien entendu, les modèles de Γ sont exactement les modèles de $\Gamma \cup \{p \Rightarrow s\}$.

La Proposition 2.34 implique également une méthode de simplification d'un ensemble de formules : si Γ contient une formule φ conséquence logique de $\Gamma - \{\varphi\}$, alors φ peut être retirée de l'ensemble de contraintes Γ sans en modifier la sémantique, $\mathsf{mod}(\Gamma) = \mathsf{mod}(\Gamma - \{\varphi\})$. L'exemple suivant éclaircit cette méthode.

Exemple 2.36. Avec cet exemple, nous allons tirer avantage des propositions et remarques précédentes pour résoudre un ensemble de contraintes ayant une certaine complexité.

On dispose de 4 variables propositionnelles, p_A, p_B, p_C, p_D, qui obéissent aux contraintes suivantes :

φ_1 : $p_B \wedge \neg p_C$
φ_2 : $p_A \Rightarrow (p_C \vee p_D)$
φ_3 : $\neg p_C \wedge (p_B \vee p_A)$

Soit $\Gamma_1 = \{\varphi_1, \varphi_2, \varphi_3\}$, cet ensemble forme l'ensemble des prémisses à partir desquelles nous allons essayer de déduire les valeurs que les variables propositionnelles peuvent prendre.

On se pose les questions suivantes :

1. *Peut-on simplifier l'ensemble Γ_1 de façon à ne pas changer l'ensemble de ses modèles, et donc de ses conséquences ?*
   (a) On remarque que la contrainte φ_3 est une **conséquence logique** de la contrainte φ_1.
   En effet, pour toute valuation v,
   — v satisfait φ_1 ssi $v(p_C) = 0$ et $v(p_B) = 1$,
   — v satisfait φ_3 ssi $v(p_C) = 0$ et $(v(p_A) = 1$ ou $v(p_B) = 1)$.
   D'où, $\mathsf{mod}(\varphi_1) \subseteq \mathsf{mod}(\varphi_3)$.
   (b) Soit $\Gamma_2 = \{\varphi_1, \varphi_2\}$, on a $\mathsf{mod}(\Gamma_2) = \mathsf{mod}(\Gamma_1)$. En effet, par définition,
   $\mathsf{mod}(\Gamma_1) = \mathsf{mod}(\varphi_1)\cap\mathsf{mod}(\varphi_2)\cap\mathsf{mod}(\varphi_3) = \mathsf{mod}(\varphi_1)\cap\mathsf{mod}(\varphi_2) = \mathsf{mod}(\Gamma_2)$.
   Donc les conséquences de Γ_1 et Γ_2 sont les mêmes et on peut alors simplifier Γ_1 par Γ_2.
2. *Quel est l'ensemble des modèles de Γ_2 ?*

Modèles de φ_1 :

p_A	p_B	p_C	p_D
0	1	0	0
0	1	0	1
1	1	0	0
1	1	0	1

Modèles de φ_2 :

p_A	p_B	p_C	p_D
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	1
1	1	1	0
1	1	1	1

L'ensemble M des modèles de Γ_2 est l'intersections des modèles de φ_1 et φ_2 :

p_A	p_B	p_C	p_D
0	1	0	1
1	1	0	1

3. Γ_2 *est-il consistant ? contradictoire ?*
   Γ_2 admet un modèle, il est donc consistant et non contradictoire.
4. *Quelles conséquences logiques pouvons nous tirer de l'ensemble* Γ_2 *?*
   Les conséquences logiques de Γ_2 sont toutes les formules dont l'ensemble des modèles contient M. On a donc entre autres :
   $\neg p_C$, p_B, $p_B \wedge \neg p_C \in \text{cons}(\Gamma)$
5. Ajoutons maintenant une nouvelle contrainte : $\Gamma_3 = \{\neg p_B \wedge \neg p_D\} \cup \Gamma_2$. On a $\mathsf{mod}(\Gamma_3) = \mathsf{mod}(\Gamma_2) \cap \mathsf{mod}(\neg p_B \wedge \neg p_D) = \varnothing$. Donc $\mathsf{mod}(\Gamma_3) = \varnothing$ et Γ_3 est contradictoire.

2.3.2.1 Compacité

Le théorème de compacité sert à caractériser la conséquence logique dans les cas où l'ensemble des formules est infini en ne considérant que des sous-ensembles finis. Par ailleurs, ce théorème jouera un rôle cruciale plus tard, dans le cadre de la preuve de complètude pour la calcul de la résolution (Théorème 3.66).

Le théorème de compacité Pour commencer, nous avons besoin du lemme suivant appelé Lemme de König.

Lemme 2.37 (König)**.** *Tout arbre infini à branchement fini possède une branche infinie.*

Démonstration. Supposons que T, qui est à branchement fini, soit infini. On définit une branche infinie dans T, ce qui mènera à la conclusion. Pour construire cette branche, on montre, par induction sur les entiers, que la propriété suivante :

$$P(n) ::= \text{ il existe une branche } e_0, \ldots, e_n \text{ telle que le sous arbre issu de } e_n \text{ est infini}$$

est vraie de tout entier $n \geq 0$.

(Base de l'induction). Pour $n = 0$, on choisit $e_0 = r$ (la racine de l'arbre) qui par hypothèse est la racine d'un arbre infini.

(Étape inductive). On suppose $P(n)$ vraie et on montre $P(n+1)$. Par hypothèse, il existe une branche $e_0, \ldots, e_n$ telle que le sous arbre issu de e_n est infini. Considérons les successeurs immédiats de e_n, disons $e_{n_1}, \ldots, e_{n_k}$: si tous étaient racines de sous-arbres finis, disons de cardinaux $p_1, \ldots, p_k$, alors il en serait de même de e_n (avec un cardinal d'au plus $p_1 + \ldots + p_{k+1}$), contradiction. Donc l'un d'entre eux est le e_{n+1} recherché. □

Nous aurons besoin du Lemme dans la forme suivante :

Lemme 2.38. *Tout arbre a branchement fini et dont toutes les branches sont finies, est fini.*

Théorème 2.39 (Compacité)**.** *Un ensemble de formules propositionnelles Γ est satisfaisable ssi tout sous-ensemble fini de Γ est satisfaisable.*

Par contraposée, le Théorème de compacité peut s'énoncer de la façon suivante :

Théorème 2.40 (Compacité)**.** *Un ensemble de formules propositionnelles Γ est contradictoire si, et seulement si, il existe un sous-ensemble fini de Γ contradictoire.*

Remarquons que l'implication « si un sous-ensemble fini de Γ est contradictoire, alors Γ est contradictoire » est trivialement vraie. Nous nous limiterons à prouver l'implication inverse.

Démonstration. On fait la preuve dans le cas où $\textsc{Prop} = \{p_0, p_1, p_2, \ldots, p_n, \ldots\}$ est un ensemble dénombrable.

Nous avons besoin d'une construction importante appelée « arbre sémantique » ou « arbre de Herbrand ». L'arbre sémantique associé est un arbre binaire infini dont toutes les arêtes à gauches sont étiquetées par 0 (le « faux ») et celles à droites sont étiquetées par 1 (le « vrai »). Chaque niveau de l'arbre est associé à une proposition. La racine (le niveau 0) est associé à p_0 : chaque fois que l'on descend d'un noeud de niveau i, ceci revient à poser p_i faux si l'on descend à gauche, et p_i vrai si l'on descend à droite. Remarquons que :

1. chaque chemin infini partant de la racine correspond à une valuation de l'ensemble des propositions ;
2. chaque noeud e à profondeur n correspond à une valuation v_e des variables $\{p_0, \ldots p_{n-1}\}$.

Nous appelons un noeud e de l'árbre *noeud d'échec* (par rapport à Γ) s'il existe une formule $\varphi_e \in \Gamma$ telle que $\textsc{Prop}(\varphi_e) \subseteq \{ p_0, \ldots p_{n-1} \}$ et $v_e(\varphi_e) = 0$, où n est la profondeur du noeud e.

On suppose que Γ est inconsistante et on montre qu'il existe un sous-ensemble $\Gamma_0 \subseteq \Gamma$ fini et inconsistant.

On commence par remarquer que chaque branche contient un noeud d'échec. En effet, si une branche n'en contient pas, elle définit un modèle de Γ, ce qui est contradictoire à l'hypothèse.

On peut donc faire la construction suivante : prenons le premier noeud d'échec de chaque branche et étiquetons ce noeud par une formule de Γ fausse sur ce noeud, puis coupons l'arbre au niveau du noeud d'echec. (Plus formellement, si π est une branche de l'arbre sémantique, dénotons par $e(\pi)$ le premier noeud d'échec sur cette branche, et choisissons $\varphi_{e(\pi)} \in \Gamma$ tel que $v_{e(\pi)}(\varphi_{e(\pi)}) = 0$; ensuite, coupons l'arbre de façon que les noeuds $e(\pi)$ deviennent des feuilles de l'arbre.)

L'arbre obtenu en tronquant ainsi toutes les branches est un arbre à branchement fini, ses branches sont finies, et donc il est fini par le Lemme de König. Le sous-ensemble $\Gamma_0 = \{ \varphi_{e(\pi)} \mid \pi \text{ une branche de l'arbre sémantique} \}$ des formules de Γ étiquetant les feuilles de l'arbre est donc fini. Or toutes les feuilles de l'arbre sont des noeuds d'échec et donc chacune des valuations rend fausse au moins une des formules de Γ_0. (Si $v \in \mathsf{Val}$, alors $v = v_\pi$ pour une branche π de l'arbre, et donc $v(\varphi_{e(\pi)}) = v_{e(\pi)}(\varphi_{e(\pi)}) = 0$.) L'ensemble $\Gamma_0 \subseteq \Gamma$ est donc fini et inconsistant. □

Exemple 2.41. Supposons que Γ soit de la forme $\{ p_0 \wedge \neg p_1, p_0 \Rightarrow p_1, \ldots \}$. Alors le noeud $e = 10$ de l'arbre sémantique est un noeud d'échec par rapport à cet ensemble Γ, car $p_0 \Rightarrow p_1 \in \Gamma$ and $v_e(p_0 \Rightarrow p_1) = 0$. (Le debut de) l'arbre sémantique, avec le noeud 10 étiqueté par la formule témoignant son échec, est représenté en Figure 2.3.

Remarque 2.42. On peut donner une preuve plus simple du théorème de compacité en utilisant les propriétés des systèmes de preuves. Nous la donnerons par la suite, lorsque nous aurons les outils nécessaires.

Corollaire du théorème de compacité :

Corollaire 2.43. *Une formule φ est conséquence d'un ensemble de formules Γ si et seulement s'il existe un sous-ensemble fini Γ_{fini} de Γ tel que $\Gamma_{fini} \models \varphi$.*

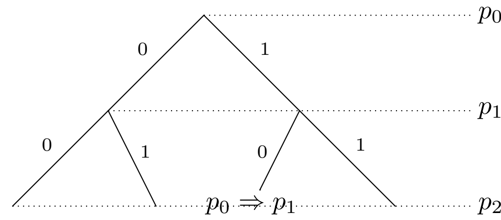


FIGURE 2.3 – Début de l'arbre sémantique avec le noeud d'échec 10 étiqueté

Démonstration.

$$
\begin{array}{lll}
\Gamma \models \varphi & \text{ssi } \Gamma \cup \{\neg\varphi\} \text{ est contradictoire} & \text{par la Proposition 2.30} \\
& \text{ssi il existe } \Gamma_f \subseteq \Gamma \cup \{\neg\varphi\} \text{ fini et contradictoire} & \text{par le Théorème de compacité} \\
& \text{ssi il existe } \Gamma_f \subseteq \Gamma \text{ fini tel que } \Gamma_f \cup \{\neg\varphi\} \text{ est contradictoire} & \\
& \text{ssi il existe un sous-ensemble fini } \Gamma_f \subseteq \Gamma \text{ tel que } \Gamma_f \models \varphi\,. & \square
\end{array}
$$

2.3.3 Décidabilité du calcul propositionnel

Une logique est décidable s'il existe un algorithme (calcul réalisable sur un ordinateur qui termine toujours pour toute donnée) qui permet de savoir pour chaque formule si elle est une tautologie (i.e. si $\models \varphi$) ou pas.

Théorème 2.44. *Le calcul propositionnel est décidable.*

Démonstration. Méthode des tables de vérité : calculer la table de vérité prenant en argument les symboles propositionnels de φ et calculer pour chaque valuation possible la valeur de φ.

Coût : $O(2^n)$ avec n la taille de φ (nombre de propositions). □

Nous verrons par la suite qu'il y a de meilleurs algorithmes, mais qu'il ont tous un coût exponentiel. La plupart de ces algorithmes débutent par une première phase de normalisation de la formule, c'est à dire qu'on modifie la syntaxe de la formule de manière à la mettre sous une forme normalisée, tout en conservant la sémantique de la formule, c'est-à-dire l'ensemble de ses modèles.

2.4 Equivalence entre formules

Il est courant de souhaiter modifier une formule, de façon à rendre son expression plus simple, ou plus facile à manipuler, et ceci en gardant bien sûr la sémantique de la formule, c'est-à-dire, sans modifier l'ensemble de ses modèles.

2.4.1 La substitution

La substitution d'une formule ψ par une formule ψ' dans une troisième formule φ (notée $\varphi_{[\psi\leftarrow\psi']}$) consiste à remplacer chaque occurrence de ψ dans φ par ψ'.

Prenons par exemple $\varphi = (\neg p \vee q) \wedge (\neg p \vee \neg r)$, $\psi = \neg p$ et $\psi' = q \Rightarrow p$. Alors $\varphi_{[\psi\leftarrow\psi']} = ((q \Rightarrow p) \vee q) \wedge ((q \Rightarrow p) \vee \neg r)$.

Plus formellement, la substitution est définie de la façon suivante :

Définition 2.45 (Substitution)**.** Soient φ, ψ, et ψ' trois formules du calcul propositionnel,

- si ψ n'est pas une sous-formule de φ, alors $\varphi_{[\psi\leftarrow\psi']} = \varphi$
- sinon si $\varphi = \psi$ alors $\varphi_{[\psi\leftarrow\psi']} = \psi'$
- sinon
  - si $\varphi = \neg\varphi'$ alors $\varphi_{[\psi\leftarrow\psi']} = \neg(\varphi'_{[\psi\leftarrow\psi']})$
  - si $\varphi = \varphi_1 \circ \varphi_2$ (où $\circ$ est un connecteurs $\wedge$, $\vee$, $\Rightarrow$) alors $\varphi_{[\psi\leftarrow\psi']} = \varphi_{1[\psi\leftarrow\psi']} \circ \varphi_{2[\psi\leftarrow\psi']}$.

Proposition 2.46. *Soient φ, ψ, et ψ' trois formules du calcul propositionnel, si $\psi \equiv \psi'$ alors $\varphi \equiv \varphi_{[\psi\leftarrow\psi']}$.*

Démonstration. Voir TD 3. □

Attention, dans le cas général, la substitution ne conserve pas la sémantique de la formule. Par exemple, p et $q \wedge \neg q$ ne sont pas équivalentes ; ainsi, les formules p et $p[p \leftarrow q \wedge \neg q]$ ne sont pas équivalentes.

2.4.2 Equivalences classiques

Nous avons vu que remplacer une sous-formule ψ d'une formule φ par une formule équivalente ψ' donne une formule notée $\varphi[\psi \leftarrow \psi']$ équivalente à φ. C'est-à-dire que cette substitution préserve les modèles, *i.e.*, $\mathsf{mod}(\varphi) = \mathsf{mod}(\varphi[\psi \leftarrow \psi'])$.

Voici quelques règles d'équivalences courantes, qui permettent de telles substitutions.

$\varphi \wedge \psi \equiv \psi \wedge \varphi$	$\varphi \vee \psi \equiv \psi \vee \varphi$	(Commutativité)
$\varphi \wedge (\psi_1 \wedge \psi_2) \equiv (\varphi \wedge \psi_1) \wedge \psi_2$	$\varphi \vee (\psi_1 \vee \psi_2) \equiv (\varphi \vee \psi_1) \vee \psi_2$	(Associativité)
$\top \wedge \varphi \equiv \varphi \wedge \top \equiv \varphi$	$\bot \vee \varphi \equiv \varphi \vee \bot \equiv \varphi$	(Élements neutres)
$\varphi \wedge \varphi \equiv \varphi$	$\varphi \vee \varphi \equiv \varphi$	(Idempotence)
$\varphi \wedge (\varphi \vee \psi) \equiv \varphi$	$\varphi \vee (\varphi \wedge \psi) \equiv \varphi$	(Absorption)
$\varphi \wedge \bot \equiv \bot \wedge \varphi \equiv \bot$	$\varphi \vee \top \equiv \top \vee \varphi \equiv \top$	(Elément absorbant)
$\varphi \wedge (\psi_1 \vee \psi_2) \equiv (\varphi \wedge \psi_1) \vee (\varphi \wedge \psi_2)$	$\varphi \vee (\psi_1 \wedge \psi_2) \equiv (\varphi \vee \psi_1) \wedge (\varphi \vee \psi_2)$	(Distributivité)
$\varphi \wedge \neg\varphi \equiv \neg\varphi \wedge \varphi \equiv \bot$	$\varphi \vee \neg\varphi \equiv \neg\varphi \vee \varphi \equiv \top$	(Complément)
$\neg\neg\varphi \equiv \varphi$		(Involution)
$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$	$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$	(Lois de De Morgan)
$\varphi \Rightarrow \psi \equiv \neg\varphi \vee \psi \equiv \neg(\varphi \wedge \neg\psi)$		(Implication matérielle)
$\varphi \Rightarrow \psi \equiv \neg\psi \Rightarrow \neg\varphi$		(Contraposition)
$\varphi_1 \Rightarrow (\varphi_2 \Rightarrow \varphi_3) \equiv (\varphi_1 \wedge \varphi_2) \Rightarrow \varphi_3$		(Curryfication)

Nous observons ici que la formule suivante :

$$[(\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_3)] \Rightarrow [\varphi_1 \Rightarrow \varphi_3] \qquad \text{(Transitivité)}$$

est une tautologie, et que $\varphi \equiv \top$ si et seulement si φ est une tautologie.

2.4.3 Formes normales

La mise sous forme normale transforme une formule en une formule équivalente (que l'on dit « normalisée ») plus adaptée au traitement algorithmique.

Notations. Remarquons que, à cause des lois d'associativité et commutativité, toutes les possibles formules construites à partir des formules $\varphi_1, \ldots, \varphi_n$ via l'application du connecteur logique $\wedge$, sont équivalentes. Par exemple

$$((\varphi_1 \wedge \varphi_2) \wedge \varphi_3) \wedge \varphi_4\,, \quad \varphi_1 \wedge ((\varphi_2 \wedge \varphi_3) \wedge \varphi_4)\,, \quad \varphi_1 \wedge (\varphi_2 \wedge (\varphi_3 \wedge \varphi_4))\,, \quad (\varphi_2 \wedge \varphi_4) \wedge (\varphi_3 \wedge \varphi_1)\,,$$

sont des formules équivalentes. Une remarque analogue vaut pour le connecteur logique $\vee$.

Ainsi, si $\varphi_1, \ldots, \varphi_n$ sont des formules, nous pourrons utiliser les notations :

$$\bigwedge_{i=1}^{n} \varphi_i = \varphi_1 \wedge \ldots \wedge \varphi_n \qquad \text{et} \qquad \bigvee_{i=1}^{n} \varphi_i = \varphi_1 \vee \ldots \vee \varphi_n$$

pour dénoter un parenthésage arbitraire de $\varphi_1 \wedge \ldots \wedge \varphi_n$ (resp. $\varphi_1 \vee \ldots \vee \varphi_n$). Les choix de l'ordre et du parenthésage ne sont donc pas significatifs, au moins du point de vue sémantique.

Nous pouvons même étendre nous considérations plus loin : si ψ possède plus que deux occurrence dans la liste $\varphi_1, \ldots, \psi_n$, nous pouvons effacer les doublons de cette liste pour obtenir des formules équivalentes. Par exemple, les formules

$$\varphi_1 \wedge \varphi_2 \wedge \varphi_1 \wedge \varphi_3\,, \quad \varphi_1 \wedge \varphi_2 \wedge \varphi_3\,,$$

sont équivalents. Moralité : si ni l'ordre, ni le parenthésage, ni la multiplicité comptes, ce qui compte dans une telle formule est sa structure d'ensemble.

Nous allons donc considérer des conjonctions et disjonctions de liste de formules (avec ou sans répétions) ; n, la longuer de la liste pourra aussi prendre les valeurs 0 et 1, en posant :

$$\bigwedge_{i=1}^{0} \varphi_i := \top\,, \qquad \bigvee_{i=1}^{0} \varphi_i := \bot\,, \qquad \text{pour } n = 0,$$

$$\bigwedge_{i=1}^{1} \varphi_i := \varphi_1\,, \qquad \bigvee_{i=1}^{1} \varphi_i := \varphi_1\,, \qquad \text{pour } n = 1.$$

Définition 2.47 (Littéral)**.** Un **littéral** est une formule atomique ou la négation d'une formule atomique. Autrement dit, c'est une formule ℓ de la forme p ou $\neg p$, où p est un symbole propositionnel.

Définition 2.48 (Clause)**.** Une **clause disjonctive** est une disjonction de littéraux : $\bigvee_{i=1}^{n} \ell_i$ où les ℓ_i sont des littéraux. Une **clause conjonctive** est une conjonction de littéraux : $\bigwedge_{i=1}^{n} \ell_i$ où les ℓ_i sont des littéraux.

Définition 2.49 (Forme normale conjonctive)**.** Une **formule conjonctive** (ou formule sous **forme normale conjonctive** (FNC), ou sous **forme clausale**) est une conjonction de clauses disjonctives : $\bigwedge_{j=1}^{m} C_j$ où les C_j sont des clauses disjonctives, ou encore $\bigwedge_{j=1}^{m} \bigvee_{i=1}^{n_j} \ell_i^j$ où les ℓ_i^j sont des littéraux.

Exemple 2.50. La formule suivante est sous forme clausale :

$$(\neg p \vee q \vee r) \wedge (\neg q \vee p) \wedge s$$

Définition 2.51 (Forme normale disjonctive)**.** Une **formule disjonctive** (ou formule sous **forme normale disjonctive** (FND)) est une disjonction de clauses conjonctives : $\bigvee_{j=1}^{m} C_j$ où les C_j sont des clauses, ou encore $\bigvee_{j=1}^{m} \bigwedge_{i=1}^{n_j} \ell_i^j$, où les ℓ_i^j sont des littéraux.

Exemple 2.52. La formule suivante est sous forme normale disjonctive :

$$(\neg p \wedge q \wedge r) \vee (\neg q \wedge p) \vee s$$

La forme normale conjonctive est en général la plus adaptée lorsqu'on cherche un modèle d'une formule car il faut chercher une valuation satisfaisant chacune des clauses de la formule. Dans l'exemple 2.50, on voit que si v est un modèle, alors forcément $v(s) = 1$; pour satisfaire la deuxième clause, il faut que $v(p) = 1$ ou $v(q) = 0$, mais alors la seule façon de satisfaire la première clause est $r = 1$. Il y a donc deux modèles.

2.4.3.1 Mise sous forme clausale, en préservant l'équivalence

L'algorithme est le suivant :

Etape 1 (élimination de l'implication). Appliquer, tant que possible, la substitution suivante :

$$\varphi \Rightarrow \psi \leftarrow \neg\varphi \vee \psi \ .$$

Etape 2 (pousser la négation vers les symboles propositionnels). Appliquer, tant que possible, les substitutions suivantes (en remplaçant le membre gauche par le membre droit) :

$$\neg\neg\varphi \leftarrow \varphi \ , \qquad \neg(\varphi \wedge \psi) \leftarrow \neg\varphi \vee \neg\psi \ , \qquad \neg(\varphi \vee \psi) \leftarrow \neg\varphi \wedge \neg\psi \ .$$

Etape 3 (pousser la disjonction vers les littéraux). Appliquer, tant que possible, les substitutions suivantes (en remplaçant le membre gauche par le membre droit) :

$$\varphi \vee (\psi_1 \wedge \psi_2) \leftarrow (\varphi \vee \psi_1) \wedge (\varphi \vee \psi_2) \ .$$

Exemple 2.53. Considérons $\varphi = (\neg(p \wedge (\neg q \vee (r \vee s)))) \wedge (p \vee q)$.
- Etape 1 : rien à faire
- Etape 2 :
  $\varphi = (\neg p \vee \neg(\neg q \vee (r \vee s))) \wedge (p \vee q)$
  $\varphi = (\neg p \vee (\neg\neg q \wedge \neg(r \vee s))) \wedge (p \vee q)$
  $\varphi = (\neg p \vee (q \wedge \neg(r \vee s))) \wedge (p \vee q)$
  $\varphi = (\neg p \vee (q \wedge \neg r \wedge \neg s)) \wedge (p \vee q)$
- Etape 3 :
  $\varphi = (\neg p \vee q) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee \neg s) \wedge (p \vee q)$

Proposition 2.54. *Le calcul précédent termine et donne une formule en forme clausale équivalente à la formule initiale.*

Remarque 2.55. Il n'y a pas unicité de la forme clausale.

2.4.3.2 Mise sous forme clausale, en préservant la satisfaisabilité

Considérons un formule du calcul propositionnel ayant ψ comme sous-formule : cette formule sera donc de la forme $\varphi[p \leftarrow \psi]$, où on peut choisir $p \notin \textsc{Prop}(\psi)$.

Proposition 2.56. *Soit $\varphi, \psi \in \mathcal{F}_{cp}$ and soit $p \notin \textsc{Prop}(\psi)$; supposons que φ ne contient pas de négations. On a alors que*

$$\mathsf{mod}(\varphi[p \leftarrow \psi]) \neq \varnothing \ \textit{ssi} \ \mathsf{mod}(\varphi \wedge (p \Rightarrow \psi)) \neq \varnothing \,.$$

C'est-à-dire, les deux formules sont equisatisfaisables.

Démonstration. Soit v d'abord une valuation telle que $v(\varphi[p \leftarrow \psi]) = 1$. On peut étendre cette valuation à p, en définissant $v(p) = v(\psi)$, et on aura alors $v(\varphi \wedge (p \Rightarrow \psi)) = 1$.

Par contre, considérons un modèle v de $\varphi \wedge p \Rightarrow \psi$ est un modèle de φ tel que $v(p) \leq v(\psi)$. Par induction sur φ, en considérant que la négation n'apparaît pas dans φ, on trouve que $1 = v(\varphi) \leq v(\varphi[p \leftarrow \psi])$. □

Exercice 2.57. Montrez que l'hypothèse que la négation n'apparaît pas dans φ est nécessaire.

Exercice 2.58. Montrez que les deux formules $\varphi[p \leftarrow \psi]$ et $\varphi \wedge (p \Rightarrow \psi)$ ne sont pas équivalentes.

Exemple 2.59. Considerez la formule

$$\bigvee_{i=1}^{n} p_{i,0} \wedge p_{i,1} . \tag{2.1}$$

L'algorithme de mise en forme normale conjonctive construit la formule

$$\bigwedge_{f:[n]\to 2} p_{1,f(1)} \vee p_{2,f(2)} \vee \ldots \vee p_{n,f(n)} .$$

Dans cet exemple le nombre de clauses produites a taille exponentielle par rapport au nombre de clauses originaires.

La Proposition 2.56 montre que la formule (2.1) est equisatisfaisable avec la formule

$$(q_1 \vee \ldots \vee q_n) \wedge \bigwedge_{i=1}^{n} (\neg q_i \vee p_{i,0}) \wedge (\neg q_i \vee p_{i,1}) ,$$

une formule en FNC, avec un nombre de clauses de taille linéaire par rapport au nombre de clauses originaires.

La Proposition 2.56 suggère donc une méthode de mise en forme clausale, préservant la satisfaisabilité, qui accélère le calcul de la forme clausale. La méthode s'applique si nous sommes intéressés seulement à l'existence d'un modéle, et non pas à énumérer tous les modèles. Parmi les défauts de la méthode, le fait qu'il introduit des nouveaux symboles propositionnels, ce qui alourdi évidemment les calculs de recherche d'un modèle.

2.5 Le problème SAT

2.5.1 Définition du problème

Définition 2.60 (Problème SAT). Le problème SAT est le problème de décision qui consiste à déterminer si $\varphi \in \mathcal{F}_{\mathrm{cp}}$ donnée en entrée admet, ou non, un modèle.

Le plus souvent, on suppose que la formule φ en entrée est en forme normale conjonctive.

La plupart des algorithmes de résolution de SAT ne se contentent pas de répondre par oui ou par non, il peuvent fournir aussi un modèle, ou même l'ensemble des modèles.

Exemple 2.61. Donnée : Une formule booléenne mise sous forme FNC :

$$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1) .$$

Question : Est-ce que la formule φ admet au moins un modèle ?

Réponse : Pour cet exemple, la réponse est oui : la valuation $v(x_1) = 0$, $v(x_2) = 1$, $v(x_3) = 1$ satisfait la formule φ, c'est-à-dire $v \in \mathsf{mod}(\varphi)$.

2.5.2 Un problème NP-complet

Théorème 2.62. *Le problème SAT est décidable.*

Démonstration. **Algorithme :** Etant donné une formule φ ayant n variables propositionnelles. Calculer les 2^n valuations possibles. Pour chacune d'entre-elles, calculer la valeur de vérité de φ. Si au moins une est vraie, alors φ est satisfaisable. □

Il existe des algorithmes plus performants, mais ces améliorations ne changent pas fondamentalement la difficulté du problème. On est devant la situation suivante. Étant donnée une formule φ, on se demande si φ admet un modèle ou non, i.e., s'il existe des valeurs de vérité attribuables aux variables propositionnelles qui satisferait φ :

- une recherche exhaustive comme dans l'algorithme précédent peut demander jusqu'à 2^n vérifications si φ possède n variables propositionnelles. Cette démarche est dite **déterministe**, mais son temps de calcul est exponentiel.

— d'un autre côté, si φ est satisfiable, il suffit d'une vérification à faire, à savoir tester précisément la configuration qui satisfait φ. Cette vérification demande un simple calcul booléen, qui se fait en temps polynomial (essentiellement linéaire en fait). Le temps de calcul cesse donc d'être exponentiel, à condition de savoir quelle configuration tester. Celle-ci pourrait par exemple être donnée par un être omniscient auquel on ne ferait pas totalement confiance. Une telle démarche est dite **non déterministe**.

La question de la satisfiabilité de φ, ainsi que tous les problèmes qui se résolvent suivant la méthode que nous venons d'esquisser, sont dits NP (pour polynomial non déterministe). Par exemple, tester si la formule φ est une tautologie équivaut, par des calculs très simples en temps polynomial, à tester que sa négation n'est pas satisfaisable (par la Proposition 2.22).

Le problème SAT joue un rôle fondamental en théorie de la complexité, puisqu'on peut montrer que la découverte d'un algorithme déterministe en temps polynomial pour ce problème permettrait d'en déduire des algorithmes déterministes en temps polynomial pour tous les problèmes de type NP (théorème de Cook). On dit que SAT (et donc également le problème de la non-démontrabilité d'une proposition) est un problème NP-complet.

2.5.3 Modélisation - Réduction à SAT

Bien que le problème soit très difficile, nous verrons que nous disposons de logiciels très performants permettant de résoudre le problème de satisfaction d'une formule. Il est donc important pour tout informaticien de savoir profiter de ces outils. L'étape préalable est la suivante : étant donné un problème qui peut-être apparemment complètement dissocié de la logique, réduire la résolution de ce problème à la satisfaction d'une formule du calcul propositionnel. Il est bien sûr nécessaire que la réduction elle-même soit réalisable en un temps raisonnable (en temps polynomial).

2.5.3.1 Comment modéliser

La plupart du temps, les problèmes sont énoncés en français (dans un fragment du français plus ou moins ambigu). La première étape est d'identifier les **propositions atomiques** d'un énoncé, il s'agit des plus petites briques de l'énoncé qui soient indécomposables et qui peuvent prendre la valeur vraie ou faux.

Il faut ensuite construire une formule traduisant les énoncés à partir des propositions, en utilisant les connecteurs booléens. On utilise pour cela la table de correspondance suivante

et, mais	$\wedge$
ou	$\vee$
ne pas, non	$\neg$
il n'est pas vrai que	$\neg$
si p alors q, q seulement si p	$p \Rightarrow q$
p si et seulement si q	$p \Leftrightarrow q$

Exemple 2.63 (suite)**.**

Si et seulement si : Considérons l'énoncé

« Le triangle est équilatéral si, et seulement si, il a trois coté égaux. »

et posons :
— *trois* : le triangle a trois côtés égaux
— *equi* : le triangle est équilatéral

L'enoncé se traduit par $trois \Leftrightarrow equi$. En effet, décomposons l'énoncé :
— « Le triangle est équilatéral si il a trois coté égaux » se traduit par $trois \Rightarrow equi$.
— « Le triangle est équilatéral seulement si il a trois coté égaux » signifie que si le triangle est équilatéral, alors il a forcément trois cotés égaux et se traduit donc par $equi \Rightarrow trois$.

On a donc $trois \Rightarrow equi \wedge equi \Rightarrow trois$, ce qui est équivalent à $equi \Leftrightarrow trois$.

Implication versus équivalence Il est d'usage, lors de la définition de concepts mathématiques, d'utiliser "si" à la place de "si et seulement si". Par exemple la définition du triangle équilatéral a plus souvent la forme suivante :

« Un triangle est équilatéral s'il a trois coté égaux. »

Pourtant il faut entendre un "si et seulement si". Il s'agit d'une convention à laquelle vous devez vous habituer.

Condition nécessaire et suffisante : Cette expression est une autre version du "si et seulement si". Posons :
— *note* : avoir une bonne note
— *travail* : travailler

et considérons l'énoncé :

« Pour avoir une bonne note, il faut et il suffit de travailler »

ou de façon équivalente :

« Pour avoir une bonne note, il est nécessaire et suffisant de travailler »

Décomposons l'énoncé :

— P_1 = « Pour avoir une bonne note il faut travailler » signifie qu'il faut nécessairement travailler pour avoir une bonne note. On a donc la table suivante :

note	*travail*	P_1
1	1	1
1	0	0
0	0	1
0	1	1

Donc $P_1 \equiv note \Rightarrow travail$.

— P_2 = « Pour avoir une bonne note, il suffit de travailler » signifie que si on travaille, on a une bonne note :

note	*travail*	P_1
1	1	1
1	0	1
0	0	1
0	1	0

Donc $P_2 \equiv travail \Rightarrow note$.

2.5.3.2 Formalisation du problème du Sudoku pour la réduction à SAT

Une grille de Sudoku est composée de $n = \ell^2$ cases, et cette grille est elle-même divisée en ℓ sous-grilles. Généralement, on prend $n = 9$ et $\ell = 3$). Au départ certaines cases sont remplies par des chiffres et d'autres sont vides. Le but du jeu est de remplir les cases vides en respectant les règles suivantes :

— On ne doit pas avoir deux chiffres identiques sur une même ligne ;
— On ne doit pas avoir deux chiffres identiques sur une même colonne ;
— Il ne doit pas y avoir deux chiffres identiques dans l'une des sous-grilles de taille $\ell * \ell$.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Pour être un Sudoku, **la grille doit accepter une et une seule solution**.

Formalisation. Pour représenter le problème sous forme de problème SAT, nous avons besoin de beaucoup de variables propositionnelles. En effet il faut n^3 variables V_{xyz} avec x, y et z *in* $[1, n]$. La variable V_{xyz} sera vraie si et seulement si la case (x, y) contient la valeur z. Par exemple, si V_{135} est vraie, alors la case $(1, 3)$ contient 5. Si on prend $n = 4$, il faudra 64 variables, pour $n = 9$ il en faut 729. Voici la formulation des règles de remplissage d'un Sudoku.

« Chaque case contient un et un seul chiffre » : pour tout $i \in [1, n]$, pour tout $j \in [1, n]$, il existe $k \in [1, n]$ tel que la case (i, j) contient k et pour tout $k' \neq k$ dans $[1, k]$: *la case* (i, j) *ne contient pas* k'.

$$A = \bigwedge_{i\in[1,n]} \bigwedge_{j\in[1,n]} \bigvee_{k\in[1,n]} \left(V_{ijk} \wedge \bigwedge_{\substack{k' \in [1,k] \\ k' \neq k}} \neg V_{i,j,k'} \right)$$

« On ne doit pas avoir deux chiffres identiques sur une même colonne » : pour toute colonne $i \in [1,n]$, pour toutes lignes $j \neq j'$ dans $[1,n]$, pour toute valeur $k \in [1,n]$: *si la case (i,j) contient k, alors la case (i,j') ne contient pas k.*

$$B = \bigwedge_{i\in[1,n]} \bigwedge_{j\in[1,n]} \bigwedge_{\substack{j' \in [1,n] \\ j' \neq j}} \bigwedge_{k\in[1,n]} (V_{i,j,k} \Rightarrow \neg V_{i,j',k})$$

« On ne doit pas avoir deux chiffres identiques sur une même ligne » : pour toute ligne $j \in [1,n]$, pour toutes colonnes $i \neq i'$ dans $[1,n]$, pour toute valeur $k \in [1,n]$: *si la case (i,j) contient k, alors la case (i',j) ne contient pas k.*

$$C = \bigwedge_{j\in[1,n]} \bigwedge_{i\in[1,n]} \bigwedge_{\substack{i' \in [1,n] \\ i' \neq i}} \bigwedge_{k\in[1,n]} (V_{i,j,k} \Rightarrow \neg V_{i',j,k})$$

« On ne doit pas y avoir deux chiffres identiques dans l'une des sous-grilles de taille $\ell * \ell$ » : pour tous $x, x' \in [1,\ell]$,

$$D = \bigwedge_{x,y\in[0,\ell-1]} \bigwedge_{\substack{i,i' \in [1+\ell x, \ell+\ell x] \\ i \neq i'}} \bigwedge_{\substack{j,j' \in [1+\ell y, \ell+\ell y] \\ j \neq j'}} \bigwedge_{k\in[1,n]} (V_{i,j,k} \Rightarrow \neg V_{i',j',k})$$

Pour assurer qu'une grille donnée est un Sudoku, il faut indiquer les chiffres déjà inscrits et vérifier qu'il y a une et une seule solution au problème. Par exemple, pour la grille donnée en exemple, on crée la formule :

$$E = V_{1,9,5} \wedge V_{2,9,3} \wedge V_{5,9,7} \wedge \dots$$

On cherche alors les modèles de la formule $A \wedge B \wedge C \wedge D \wedge E$. Si il n'y a qu'un seul modèle, alors la grille est un Sudoku dont la solution est donnée par ce modèle.

2.5.4 Algorithmes de résolution de SAT

De nombreux algorithmes ont été proposés pour résoudre SAT, nous en présentons quelques-uns, en nous focalisant sur le cas où la formule dont il faut décider la satisfaisabilité est déjà en forme normale conjonctive.

Nous allons donc présenter ici quelques calculs sur les formules en FNC qui nous utiliserons dans le cadres de ces algorithmes qui suivent.

Remarque 2.64. Nous pouvons identifier une formule en forme normale conjonctive φ avec l'ensemble $\mathcal{C}_\varphi$ des ses clauses, car évidemment nous avons $\mathsf{mod}(\varphi) = \mathsf{mod}(\mathcal{C}_\varphi)$, voir la Proposition 2.33. Par exemple, si

$$\varphi = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$$

(donc φ est en FNC), alors son ensemble de clauses associé est

$$\mathcal{C}_\varphi = \{ \neg x_1 \vee x_2 , \neg x_2 \vee x_3 , \neg x_3 \vee x_4 , \neg x_1 \vee \neg x_4 \} .$$

Evidemment, etant donné un ensemble de clauses $\mathcal{C}$, nous pouvons poser

$$\varphi = \bigwedge_{C \in \mathcal{C}} C$$

de facon que $\mathcal{C} = \mathcal{C}_\varphi$. Remarquons que les algorithmes manipulent plutôt des ensembles des clauses, au lieu des formules en FNC.

Remarque 2.65 (Simplifications)**.** Avant chercher un (ou plusieurs) modèle(s) d'une formule sous forme clausale (ou d'un ensemble de clauses), nous pouvons optimiser les calculs qui suivront en appliquant les règles suivantes :

Tiers exclu : Les clauses comportant deux littéraux opposés (par ex. $p \vee q \vee \neg r \vee \neg q$) sont valides (par le tiers-exclu) et peuvent donc être supprimées.

Fusion (ou factorisation) : On peut supprimer les répétitions d'un littéral au sein d'une même clause (par ex. $\neg p \vee q \vee \neg r \vee \neg p$ équivaut à $\neg p \vee q \vee \neg r$).

Subsumption : Si, dans une formule clausale, une clause C_i est incluse dans une clause C_j (c'est-à-dire, tout littéral apparaissant dans C_i apparaît aussi dans C_j, on dit alors que C_i *subsume* C_j), alors la clause C_j peut être supprimée de la forme clausale (ou de l'ensemble de clauses $\mathcal{C}$ qui represente la formule en FNC). En fait, $C_i \models C_j$ et donc $\mathsf{mod}(\mathcal{C}) = \mathsf{mod}(\mathcal{C} \setminus \{ C_j \})$, voir la Proposition 2.34. Par exemple $C_i = p \vee q \vee r$ est incluse dans $C_j = p \vee \neg s \vee t \vee q \vee r$, de façon que l'on peut supprimer C_i d'une formule clausale de la forme $C_1 \wedge \ldots C_i \wedge \ldots C_j \wedge \ldots C_n$.

Remarque 2.66 (Substitions simples)**.** Soit ψ une formule est sous forme clausale, et $\mathcal{C}$ l'ensemble de ses clauses. Le calcul d'une formule equivalente à $\psi[p \leftarrow \varphi]$, où $\varphi \in \{ \bot, \top \}$, peut se faire aisement via une procédure purement syntaxique sur l'ensemble $\mathcal{C}$. Définissons cette procedure :

$\mathcal{C}[p \leftarrow \top]$ **:** est l'ensemble obtenu de $\mathcal{C}$ en supprimant toutes les clauses contenant p, et en supprimant $\neg p$ de toutes les clauses contenant $\neg p$.

$\mathcal{C}[p \leftarrow \bot]$ **:** est l'ensemble obtenu de $\mathcal{C}$ en supprimant toutes les clauses contenant $\neg p$, et en supprimant p de toutes les clauses contenant p.

Clairement, nous avons que

$$\psi[p \leftarrow \varphi] \equiv \bigwedge \{ C \mid C \in \mathcal{C}[\leftarrow \varphi] \} .$$

Si $\ell \in \{ p, \neg p \}$ est un littéral, nous utiliserons la notation $\mathcal{C}[\ell \leftarrow \top]$ pour $\mathcal{C}[p \leftarrow \top]$ si $\ell = p$, sinon, si $\ell = \neg p$, alors cette notation sera utilisée pour $\mathcal{C}[p \leftarrow \bot]$. C'est donc la substitution qui fore ce littéral à être vrai. Nous utiliserons la notation $\mathcal{C}[\ell \leftarrow \bot]$ pour $\mathcal{C}[\overline{\ell} \leftarrow \bot]$, $\overline{\ell}$ est le littéral opposé de ℓ : $\overline{\ell} = \neg p$ si $\ell = p$, et $\overline{\ell} = p$ si $\ell = \neg p$.

e

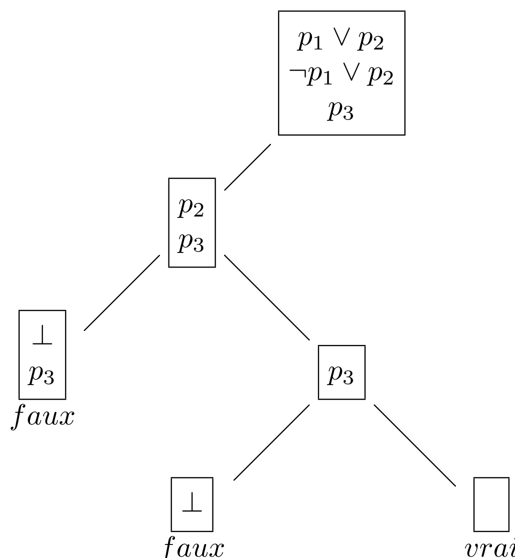
2.5.4.1 Algorithme de Quine

Nous présentons d'abord la méthode de Quine dans le cas restreint de formules mises au préalable sous forme normale conjonctive, assimilées donc à un ensemble de clauses. La discussion de la méthode de Quine nous aidera à comprendre de près le fonctionnement de l'algorithme de Davis-Putnam-Logemann-Loveland.

La méthode ne fait rien d'autre que parcourir l'arbre de toutes les solutions (l'arbre dont les branches complètes sont les valuations, appelé arbre sémantique ou arbre de Herbrand dans la preuve du Théorème 2.40). Chaque fois qu'une variable est affectée à la valeur $x \in \{ 0, 1 \}$, le calcul se fait recursivement avec $\mathcal{C}[p \leftarrow \bot]$, si $x = 0$, et avec $\mathcal{C}[p \leftarrow \top]$, sinon.

Algorithme de Quine
entrée : un ensemble de clauses $\mathcal{C}$ sortie : $vrai$ si $\mathcal{C}$ est satisfaisable ou $faux$ sinon
simplifier l'ensemble de clauses (cf. Remarque 2.65) ; si $\mathcal{C} = \varnothing$ retourner $vrai$ si $\mathcal{C}$ contient la clause $\bot$ retourner $faux$ choisir le prochain $p \in$ Prop apparaissant dans une clause si $Quine(\mathcal{C}[p \leftarrow \bot]) = vrai$ alors retourner vrai sinon retourner $Quine(\mathcal{C}[p \leftarrow \top])$

Exemple 2.67. Considerons $\{ p_1 \vee p_2, \neg p_1 \vee p_2, p_3 \}$. L'algorithme explore l'arbre de Herbrand selon un parcours en profondeur gauche comme suit :



Observons dans l'exemple précédent que nous sommes contraintes par l'algorithme de Quine à essayer d'abord p_1, ensuite p_2, et puis p_3 ; de façon similaire, l'algorithme demande d'évaluer un symbole propositionnel d'abord à faux, et puis à vrai. Cette stratégie nous amène le plus souvent à des calculs inutiles : par exemple, dans l'exemple précédent, nous construisons des noeuds suite aux évaluations de p_2 et p_3 à faux, quand il est tout à fait évident que ces évaluations nous amènerons à un échec.

En principe, nous ne sommes pas obligés à suivre un ordre fixé au début. Un algorithme pourra donc essayer d'améliorer la performance l'algorithme de Quine, en construisant à la volée un ordre d'exploration des symboles propositionnels et des affectations des valeurs de vérité, qui soit optimisé pour l'ensemble des clauses passé en entrée.

2.5.4.2 Algorithme de Davis-Putnam-Logemann-Loveland

L'algorithme DPLL est considéré, à ce jour, comme l'une des méthodes les plus efficaces parmi celles permettant de résoudre le problème SAT ; la plus part des outils des solutions des contraintes (« SAT solvers », en anglais) sont implémentent une variante de cet algorithme. Il peut être vu comme un raffinement de la méthode de Quine ; l'amélioration principale qu'elle apporte est

— la réduction du branchement via la propagation des clauses unitaires ;

— l'utilisation d'heuristiques pour accélérer le parcours des solutions.

Algorithme $DPLL$
entrée : un ensemble de clauses $\mathcal{C}$ sortie : $vrai$ si $\mathcal{C}$ est satisfaisable ou $faux$ sinon
simplifier l'ensemble de clauses (cf. Remarque 2.65) ; si $\mathcal{C} = \varnothing$ retourner $vrai$ si $\mathcal{C}$ contient la clause $\bot$ retourner $faux$ si $\mathcal{C}$ contient la clause p retourner $DPLL(\mathcal{C}[p \leftarrow \top])$ (propagation des si $\mathcal{C}$ contient la clause $\neg p$ retourner $DPLL(\mathcal{C}[p \leftarrow \bot])$ clauses unitaires) choisir un littéral ℓ depuis une clause, **avec la bonne heuristique !** si $DPLL(\mathcal{C}[\ell \leftarrow \top]) = vrai$ alors retourner vrai sinon retourner $DPLL(\mathcal{C}[\ell \leftarrow \bot])$

Heuristiques. Les heuristiques sont très importantes car elles permettent de réduire rapidement la taille de l'arbre de recherche. Parmi les heuristiques possibles :

Littéraux purs. Choisir les littéraux qui n'apparaissent que positivement (resp. négativement) et, parmi ceux-ci, choisir ceux qui sont présents le plus souvent.

Fréquence des variables. Choisir les symboles propositionnels qui apparaissent le plus, dans les clauses les plus courtes.

Littéraux "courts". Choisir les littéraux apparaissant les plus, dans les clauses les plus courtes.

Les heuristiques précédentes peuvent se formaliser en définissant des fonctions qui donnent une pondération à chaque littéral. Par exemple, pour donner un sens à l'heuristique « choisir un symbole propositionnel apparaissant le plus, dans les clauses les plus courtes », nous pouvons définir, pour un littéral ℓ,

$$\mathrm{rank}_{\mathtt{lit}}(\ell, \mathcal{C}) = \sum_{\ell \in C, C \in \mathcal{C}} \frac{1}{\mathrm{card} C},$$

et, pour un symbole propositionnel p,

$$\mathrm{rank}_{\mathtt{prsym}}(p, \mathcal{C}) = \mathrm{rank}_{\mathtt{lit}}(p, \mathcal{C}) + \mathrm{rank}_{\mathtt{lit}}(\neg p, \mathcal{C}),$$

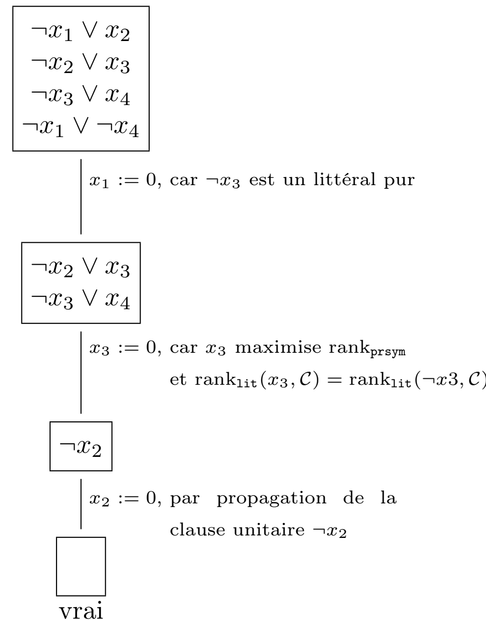
et ainsi choisir un symbole propositionnel variable qui maximise la fonction $\mathrm{rank}_{\mathtt{prsym}}$. Ensuite, nous pouvons choisir le littéral p si $\mathrm{rank}_{\mathtt{lit}}(p, \mathcal{C}) > \mathrm{rank}_{\mathtt{lit}}(\neg p, \mathcal{C})$, et $\neg p$ sinon.

Notez que la définition que nous avons proposé n'est la seule. Par exemple, nous aurions pu définir

$$\mathrm{rank}_{\mathtt{lit}}(\ell, \mathcal{C}) = \alpha \cdot \mathrm{card}\{\, C \mid \ell \in C \,\} + \beta \cdot \frac{\mathrm{card}\{\, C \mid \ell \in C \,\}}{\sum_{\ell \in C} \mathrm{card} C},$$

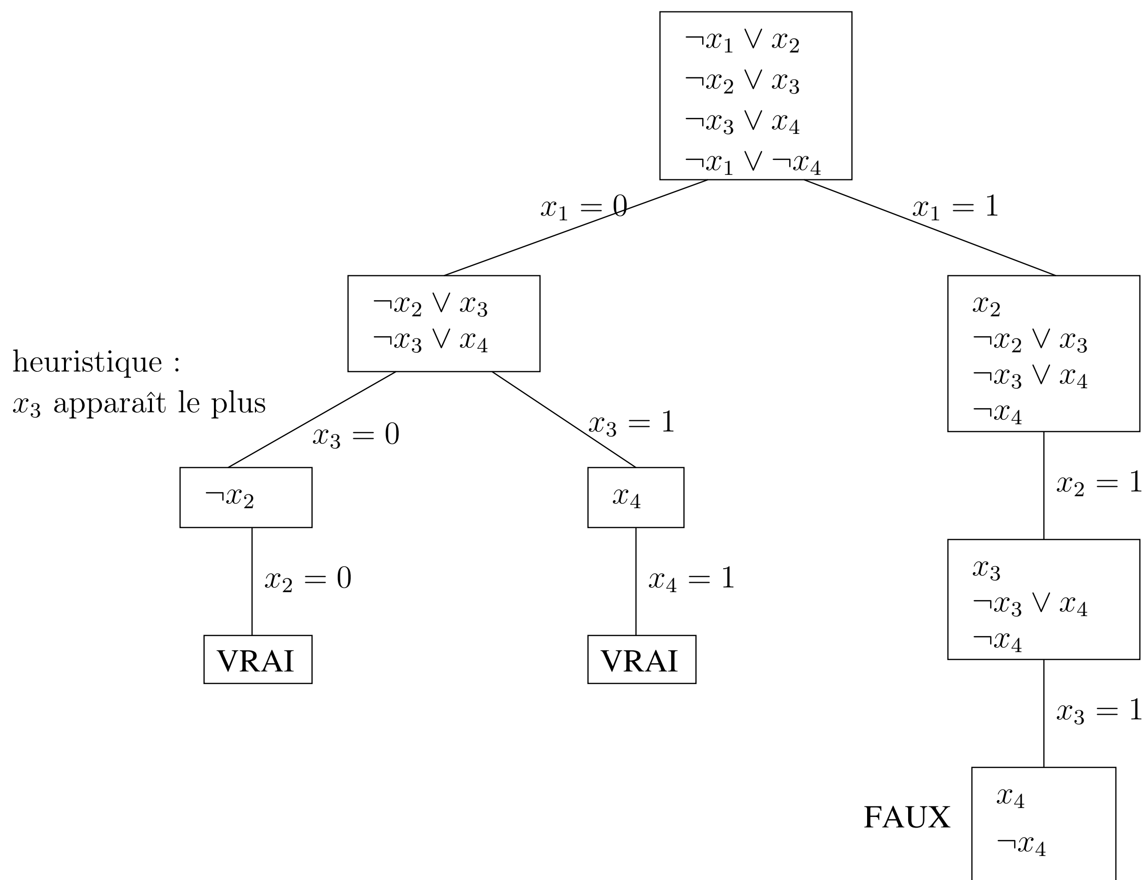
pour des pondérations α et β bien choisies.

Exemple 2.68. Considérons l'ensemble de clauses $\mathcal{C} = \{\, \neg x_1 \vee x_2 \,,\, \neg x_2 \vee x_3 \,,\, \neg x_3 \vee x_4 \,,\, \neg x_1 \vee \neg x_4 \,\}$. Lálgorthme DPLL construit lárbre suivant :



La valeur de vérité de x_4 n'est pas affecté sur cette branche : cela veut dire que chaque valeur pour x_4 donne un modèle, si $v(x_1) = v(x_3) = v(x_2) = 0$.

Exemple 2.69. Nous pouvons modifier l'algorithme DPLL (et bien sur, l'algorithme de Quine aussi), afin qu'il trouve tous les modèles d'un ensemble de clauses. Avec le même $\mathcal{C}$ de l'exemple précédent, l'algorithme produira le parcours suivant :



Les modèles de cette formule sont donc les suivants :

x_1	x_3	x_2	x_4
0	0	0	0
0	0	0	1
0	1	0	1
0	1	1	1

On remarque dans cet exemple que :

— dans la branche de droite, la propagation des clauses unitaires permet de ne suivre qu'un seul chemin ;

— dans la branche de gauche, l'heuristique, faisant choisir x_3 plutôt que x_2 ou x_4, réduit l'exploration car elle produit plus vite des clauses unitaires.

L'exécution est meilleure que si on avait fait une simple exploration de toutes les solutions (Algorithme de Quine).

Exercice 2.70. Montrez que, pour tout $n \geq 1$, il existe un ensemble de clauses $\mathcal{C}_n$, dont les symboles propositionnels sont $\{ p_1, \ldots, p_n \}$ tel que l'arbre construit par l'algorithme de Quine est un arbre complet de profondeur n (donc, avec 2^n noeuds) où, par contre, l'arbre exploré par l'algorithme DPLL est une branche de longueur n (donc, seulement $n+1$ noeuds sont explorés par l'algorithme).

2.5.4.2.1 Lecture conseillées. L'implantation efficace de l'algorithme DPLL peut être assez subtile. Par exemple, l'heuristique des littéraux purs se révèle (à l'état actuel de connaissances)

assez coûteuse ; pour cette raison, on préfère ne pas la mettre en ouvre. Le lecteur curieux pourra approfondir le sujet en lisant [ES04].

2.5.4.3 Algorithmes incomplets

Il existe aussi des semi-algorithmes (ou algorithmes incomplets) très performants, ceux-ci ne parcourent pas l'ensemble des solutions et peuvent donc ne pas répondre. En général, il ne répondent pas à la question de l'insatisfaisabilité. Ils ont l'avantage de trouver souvent rapidement un modèle lorsqu'il existe, mais il n'en trouvent pas toujours (même si la formule est satisfiable). Par contre, si la formule est non-SAT, on n'obtient aucune réponse.

Algorithme de Hill Climbing (optimisation stochastique) : Au départ, on choisit une valuation aléatoire et on dispose d'un critère de qualité (par exemple le nombre de clauses non satisfaites). A chaque étape, on choisit une variable, on inverse la variable si cela améliore le critère, sinon on l'inverse avec une probabilité faible (pour éviter les optimum locaux).

Algorithmes génétiques : Ces algorithmes sont inspirés de la sélection naturelle. Au départ, on se donne aléatoirement un certain nombre de valuations qui forme la population initiale, puis à chaque étape, on fait évoluer la population par brassage génétique en essayant de tendre vers des modèles de la formule : par croisement des meilleurs individus, par mutation d'individus et élimination des solutions les plus faibles.

2.5.4.4 Conclusion

Il existe donc de nombreux algorithmes, certains sont des améliorations de ceux présentés ci-dessus, souvent par des heuristiques par exemple sur l'ordre d'utilisation des règles. Les meilleurs complexités atteintes pour ces algorithmes oscillent entre $O(1.5^n)$ et $O(1.3^n)$

2.5.5 Sous-classes de SAT

2.5.5.1 2-SAT

Le problème 2-SAT est celui de la satisfaisabilité d'une formule sous forme clausale dont les clauses sont d'ordre 2 (i.e., chaque clause est une disjonction d'au plus deux littéraux).

Exemple : $\varphi = (p_1 \vee p_2) \wedge (\neg p_1 \vee p_3) \wedge (\neg p_2 \vee p_1) \wedge (\neg p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_3)$ est sous forme clausale d'ordre deux.

Problème 2-SAT
entrée : un ensemble de clauses $\mathcal{C}$ d'ordre 2
sortie : *vrai* si $\mathcal{C}$ est satisfaisable ou *faux* sinon

Contrairement à SAT, ce problème est résolvable en temps polynomial.

Algorithme 2-SAT :
Une clause d'ordre deux $\ell_1 \vee \ell_2$ est équivalente à $(\neg\ell_1 \Rightarrow \ell_2) \wedge (\neg\ell_2 \Rightarrow \ell_1)$. Pour résoudre SAT pour une formule φ d'ordre 2, on construit le graphe orienté $G(\varphi) = (S, A)$ dual (appelé graphe 2-SAT) selon les deux règles suivantes :

- l'ensemble des sommets est $S = \{\neg p \mid p \in \textsc{Prop}(\varphi)\} \cup \textsc{Prop}(\varphi)$ (où $\textsc{Prop}(\varphi)$ est l'ensemble des variables propositionnelles de la formule φ). C'est l'ensemble des littéraux sur $\textsc{Prop}(\varphi)$
- l'ensemble des arcs est $A = \{(\ell_1, \ell_2) \mid \varphi$ contient une clause équivalente à $\ell_1 \Rightarrow \ell_2\}$.

Chaque clause $\ell_1 \vee \ell_2$ est donc associée à deux arcs $(\neg\ell_1, \ell_2)$ et $(\neg\ell_2, \ell_1)$.

Alors, la formule φ est insatisfaisable ssi il existe une variable p telle qu'il existe dans $G(\varphi)$ un chemin allant de p à $\neg p$ et un chemin allant de $\neg p$ à p. En effet, cela signifie alors que $\varphi \models (p \Rightarrow \neg p) \wedge (\neg p \Rightarrow p)$ ce qui est équivalent à $\bot$.

Une autre formulation est la suivante : la formule φ est satisfaisable si et seulement si pour chaque variable propositionnelle p, les sommets p et $\neg p$ du graphe 2-SAT sont dans deux composantes fortement connexes distinctes. (une composante fortement connexe d'un graphe orienté G

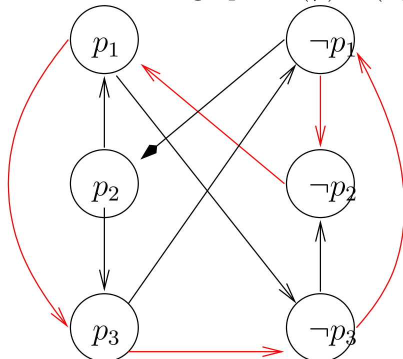
est un sous-graphe maximal de G tel que pour toute paire de sommets u et v dans ce sous-graphe, il existe un chemin de u à v et un chemin de v à u.)

L'algorithme de Tarjan permet de calculer les composantes fortement connexes d'un graphe orienté en $\mathcal{O}(|S| + |A|)$, donc 2-SAT est bien polynomial.

Exemple 2.71. $\varphi = (p_1 \vee p_2) \wedge (\neg p_1 \vee p_3) \wedge (\neg p_2 \vee p_1) \wedge (\neg p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_3)$

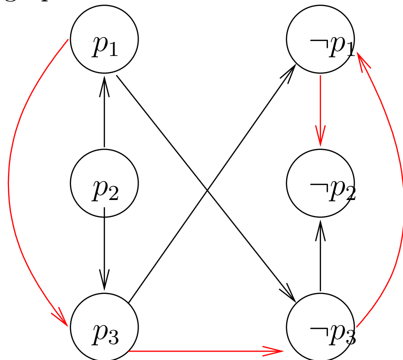
On a :
- $(p_1 \vee p_2) \equiv (\neg p_1 \Rightarrow p_2) \wedge (\neg p_2 \Rightarrow p_1)$
- $(\neg p_1 \vee p_3) \equiv (p_1 \Rightarrow p_3) \wedge (\neg p_3 \Rightarrow \neg p_1)$
- $(\neg p_2 \vee p_1) \equiv (p_2 \Rightarrow p_1) \wedge (\neg p_1 \Rightarrow \neg p_2)$
- $(\neg p_2 \vee p_3) \equiv (p_2 \Rightarrow p_3) \wedge (\neg p_3 \Rightarrow \neg p_2)$
- $(\neg p_1 \vee \neg p_3) \equiv (p_1 \Rightarrow \neg p_3) \wedge (p_3 \Rightarrow \neg p_1)$

On construit le graphe $G(\varphi) = (S, A)$:



On remarque qu'il existe un cycle passant par p_1 et $\neg p_1$, donc la formule est insatisfaisable.

Si on considère maintenant la formule $\varphi' = (\neg p_1 \vee p_3) \wedge (\neg p_2 \vee p_1) \wedge (\neg p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_3)$. Le graphe est alors le suivant :



On voit sur le graphe que $\varphi' \models p_1 \Rightarrow \neg p_1$, $\varphi' \models p_2 \Rightarrow \neg p_2$ et $\varphi' \models p_3 \Rightarrow \neg p_3$. Il n'y a donc qu'un seul modèle : $v(p_1) = v(p_2) = v(p_3) = 0$.

2.5.5.2 3-SAT

Le problème 3-SAT est lui aussi NP-complet (c'est à dire qu'il est aussi difficile que le problème SAT). Pour le démontrer, il suffit de prouver que le problème SAT est polynomialement réductible à 3-SAT : cela signifie que répondre à la question SAT revient à répondre à 3-SAT et que le temps nécessaire à transformer SAT en 3-SAT est polynomial.

Preuve : On remarque que toute clause $\varphi_n = \ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$ avec $n \geq 3$ peut se mettre sous la forme :

$\psi_n = (\ell_1 \vee \ell_2 \vee q_1) \wedge (\ell_3 \vee \neg q_1 \vee q_2) \wedge \ldots \wedge (\ell_{n-2} \vee \neg q_{n-4} \vee q_{n-3}) \wedge (\ell_{n-1} \vee \ell_n \vee \neg q_{n-3})$ où les q_i sont de nouveaux symboles propositionnels.

Donc le problème SAT est polynomialement réductible à 3-SAT. On conclut que 3-SAT est NP-complet.

Tous les problèmes n-SAT avec n supérieur ou égal à 3 sont également des problèmes NP-complets.

2.5.5.3 Horn-SAT

Une *clause de Horn* est une clause comportant au plus un littéral positif. C'est donc une disjonction de la forme $\neg p_1 \vee \ldots \vee \neg p_n \vee p$, où les p_i et p sont des variables propositionnelles. Selon qu'elles comportent ou non un littéral positif (resp. négatif), les clauses de horn sont de l'une des trois formes suivantes :

(a) $\neg p_1 \vee \ldots \vee \neg p_n \vee p$ avec $n > 0$;

(b) $\neg p_1 \vee \ldots \vee \neg p_n$ avec $n > 0$;

(c) p ;

(d) $\bot$;

Le problème Horn-SAT s'énonce de la façon suivante :

Horn-SAT
Entrée : un ensemble $\mathcal{C}$ de clauses de Horn
Question : $\mathcal{C}$ est-il satisfiable ?

Ce problème est résolvable en temps polynomial.

Algorithme : Il convient d'abord de remarquer les équivalences suivantes :
— $\neg p_1 \vee \ldots \vee \neg p_n \vee p \equiv (p_1 \wedge p_2 \wedge \ldots \wedge p_n) \Rightarrow p$ (avec $n > 0$) ;
— $\neg p_1 \vee \ldots \vee \neg p_n \equiv (p_1 \wedge p_2 \wedge \ldots \wedge p_n) \Rightarrow \bot$ avec $n > 0$;

Si q est une variable propositionnelle, et $\mathcal{C}$ est une clause de Horn, on note $\mathcal{C}/q$ la clause de Horn définie par
— $\mathcal{C}/q = (p_2 \wedge \ldots \wedge p_n) \Rightarrow p$ si $\mathcal{C} = (p_1 \wedge p_2 \wedge \ldots \wedge p_n) \Rightarrow p$, $p_1 = q$ et $n > 0$;
— $\mathcal{C}/q = vrai$ si $\mathcal{C} = (p_1 \wedge p_2 \wedge \ldots \wedge p_n) \Rightarrow p$, $p = q$ et $n > 0$;
— $\mathcal{C}/q = (p_2 \wedge \ldots \wedge p_n)$ si $\mathcal{C} = (p_1 \wedge p_2 \wedge \ldots \wedge p_n)$, $p_1 = q$ et $n > 0$;
— $\mathcal{C}/q = p$ si $\mathcal{C} = p$ et $p \neq q$
— $\mathcal{C}/q = \top$ si $\mathcal{C} = q$
— $\mathcal{C}/q = \bot$ si $\mathcal{C} = \bot$

Les clauses réduites à une variable propositionnelle sont appelés faits.

Algorithme :
Données : Un ensemble $\mathcal{C}_0$ de clauses de Horn
Sortie : $\mathcal{C}_0$ est-il satisfaisable ?

$\mathcal{C} = \mathcal{C}_0$
tant que faits($\mathcal{C}$) $\neq \varnothing$
  Choisir $p \in$ faits($\mathcal{C}$)
  $\mathcal{C} = \mathcal{C}/p$;
si $\bot \in \mathcal{C}$ **alors** Retourner "inconsistant"
sinon Retourner "satisfaisable" ;

En effet, tout ensemble de clauses ne contenant pas de faits ni la clause $\bot$ est satisfaisable : il suffit de mettre à 0 toutes les variables apparaissant dans les clauses.

2.5.6 Les SAT-solvers

Puisque le problème SAT est présent dans de nombreux domaines de l'informatique, le développement de SAT-solvers efficaces est un des défis majeur de l'informatique. De nouveaux solvers sont constamment développés pour répondre à des besoins spécifiques. Des concours mondiaux de SAT-solvers sont d'ailleurs organisés chaque année (voir `http://www.satcompetition.org`).

On peut citer parmis ces nombreux solvers : zChaff datant de 2001 qui utilise l'agorithme de Chaff qui est une amélioration de DP, Siege en 2003, MiniSAT en 2005 logiciel open-source, picoSAT dont les méthodes s'inspirent de l'algorithme de Chaff, SARzilla en 2009 qui a gagné de nombreux prix.

Enfin remarquons que les solvers modernes exploitent les techniques les plus recentes de l'informatique, comme l'apprentissage, la programmation parallèle exploitant l'architecture multicoeur des processeurs récents, etc.

2.5.7 Applications

2.5.7.1 Vérification d'un circuit logique

On encode le circuit et les entrées/sorties désirées et on teste SAT

2.5.7.2 Satisfaction de contraintes

Les problèmes de satisfaction de contraintes (CSP) sont des problèmes mathématiques où l'on cherche des états ou des objets satisfaisant un certain nombre de contraintes ou de critères. Ici on sort un peu du contexte simple de la décidabilité, il ne suffit plus de dire qu'une formule est vraie, il faut en exhiber un modèle.

Les problèmes d'emploi du temps, de coloration de graphe, ou les sudoku.

Emploi du temps Pour choisir un créneau horaire pour le cours de Logique, on doit respecter les contraintes suivantes :

- Pas en même temps que les TP de RO du M1 informatique
- Pas avant 10h30 le matin
- Pas en même temps que les autres enseignements de L3 info
- Pas en même temps que les autres enseignements obligatoires de L3 math
- Pas en même temps que tout autre cours en amphi.

2.5.7.3 Diagnostic

Le diagnostic est une discipline de l'intelligence artificielle qui vise le développement d'algorithmes permettant de déterminer si le comportement d'un système est conforme au comportement espéré. Si il ne l'est pas, il faut être capable de trouver le dysfonctionnement.

Le diagnoser doit déterminer si un système a un comportement défectueux étant donnée l'observation des entrées et sorties du système et d'observation d'états internes. Le modèle du système peut être traduit en un ensemble de contraintes (disjonctions) : pour chaque composant S du système, une variable propositionnelle $Ab(S)$ est créée qui est évaluée à vraie si le composant a un comportement anormal (Abnormal). Les observations peuvent être également traduites par un ensemble de disjonctions. L'assignation trouvée par l'algorithme de satisfaisabilité est un diagnostic.

On peut simplifier la modélisation par des formules comme les suivantes :

$\neg Ab(S) \Rightarrow Int1 \wedge Obs1$

$Ab(S) \Rightarrow Int2 \wedge Obs2$

Les formules se lisent de la manière suivante : si le système n'a pas un comportement anormal, alors il produira le comportement interne $Int1$ et le comportement observable $Obs1$. Dans le cas d'un comportement anormal, il produira le comportement interne $Int2$ et les observations $Obs2$. Étant données les observations Obs, il faut déterminer si le comportement du système est normal ou non ($\neg Ab(S)$ ou $Ab(S)$)

2.5.7.4 Planification

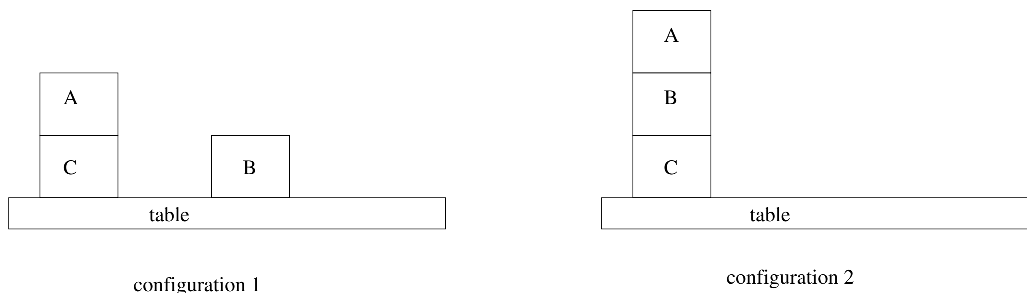
La planification est une discipline de l'intelligence artificielle qui vise le développement d'algorithmes pour produire des plans (en d'autre termes, une planification), typiquement pour l'exécution par un robot ou tout autre agent.

Un planificateur typique manipule trois entrées (toutes codées dans un langage formel qui utilise des prédicats logiques) :

- une description de l'état initial d'un monde,
- une description d'un but à atteindre et
- un ensemble d'actions possibles (parfois appelés opérateurs)

Chaque action spécifie généralement des préconditions qui doivent être présentes dans l'état actuel pour qu'elle puisse être appliquée, et des postconditions (effets sur l'état actuel). Le problème de planification classique consiste à trouver une séquence d'actions menant d'un état du système à un ensemble d'états. Par exemple, voici un problème de planification simple. On dispose de boites sur une tables, dans une configuration initiale, et on souhaite obtenir une nouvelle configuration

(par exemple passer de conf 1 de la figure à a conf 2). La seule action possible est de déplacer une boite non recouverte par une autre sur la table ou sur une autre boite.



2.5.7.5 Model checking

Le Model Checking désigne une famille de techniques de vérification automatique des systèmes dynamiques (souvent d'origine informatique ou électronique). Il s'agit de vérifier algorithmiquement si un modèle donné, le système lui-même ou une abstraction du système, satisfait une spécification, souvent formulée en termes de logique temporelle.

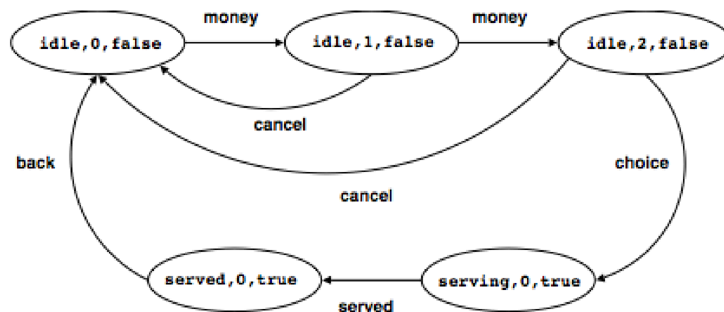
Représentation formelle : $\mathcal{M} \models^? \varphi$

— $\mathcal{M}$: (modèle du) programme sous observation

— φ : propriété à vérifier

— pré-requis : sémantique (opérationnelle), langage de spécification

Exemple : voici la modélisation à partir d'une machine à état, d'une machine à café très simplifiée.



cette machine est le modèle $\mathcal{M}$, sur lequel on veut vérifier des propriétés φ (exprimées par des formules de logique) telles que "on obtient un café seulement si on paie 2 euros".

Un exemple de model checker : POEM est un model checker développé au LIF par Peter Niebert qui utilise un SAT-solver pour finaliser ses calculs.

2.5.7.6 Cryptographie

La complexité du problème SAT est une composante essentielle de la sécurité de tout système de cryptographie.

Par exemple une fonction de hachage sécurisée constitue une boîte noire pouvant être formulée en un temps et un espace fini sous la forme d'une conjonction de clauses normales, dont les variables booléennes correspondent directement aux valeurs possibles des données d'entrée de la fonction de hachage, et chacun des bits de résultat devra répondre à un test booléen d'égalité avec les bits de données d'un bloc de données d'entrées quelconque. Les fonctions de hachages sécurisées servent notamment dans des systèmes d'authentification (connaissance de données secrètes d'entrée ayant servi à produire la valeur de hachage) et de signature (contre toute altération ou falsification "facile" des données d'entrée, qui sont connues en même temps que la fonction de hachage elle-même et de son résultat).

2.5.7.7 Bio-informatique

Certains problèmes de traitement du génome se modélisent par des formules propositionnellse et sont résolus à l'aide de SAT-solvers.

2.5.8 Sur la modélisation

Tous les applications citées ci-dessus ont un point commun : partant d'un problème qui semble complètement éloigné de la logique, on obtient une solution à ce problème en résolvant le problème SAT. La phase important de ce travail est donc la **modélisation**, qui permet par abstraction d'un objet concret d'obtenir un modèle représentant son fonctionnement et/ou ses propriétés. Dans le cas cas qui nous intéresse, le modèle est une formule du calcul propositionnel.

Ce mécanisme est très puissant, il permet de résoudre des problèmes très concrets, d'ingénierie par exemple, en utilisant des résultats théoriques existant sur des objets mathématiques.

2.6 Systèmes de preuves

2.6.1 La notion de système formel

Un *système de preuves* définit des règles de calcul entre formules qui simulent le raisonnement. Il est défini par

— le langage et les formules sur ce langage,
— les axiomes : formules supposées vraies,
— les règles d'inférence : une règles d'inférence permet de déduire une nouvelle formule, la conclusion, à partir d'un ensemble de formules, les prémisses ou hypothèses.

On note $\vdash \varphi$ si φ peut se déduire grace au système de preuves , i.e., si il est possible de l'obtenir à partir des axiomes, en appliquant les règles de réécriture du système. Soit Γ un ensemble de formules, on note $\Gamma \vdash \varphi$ si φ peut se déduire dans le système formel considéré, à partir des axiomes et des formules dans Γ.

Deux questions fondamentales sont systématiquement posées pour relier le calcul et la sémantique dans une logique : existe-il un système formel qui soit

1. **correct** : une formule déduite est une tautologie ?
2. **complet** : toute tautologie peut-elle se déduire ?

Par extension, un *système formel* est un ensemble de règles permettant d'inférer une nouvelle relation (la conclusion) à partir de relations données (le prémisses). Par exemple, le calcul des séquents est un système formel permettant de déduire une couple (Γ, Δ) (notée comme habituellement par $\Gamma \vdash \Delta$), où Γ et Δ sont des listes (resp. multi-ensembles, resp. ensembles) de formules, à partir d'autres couples du même type. Les théorèmes de correction et complétude pour le calcul des séquents montreront que le couple (Γ, Δ) est dérivable dans le calcul si est seulement si $\bigwedge_{\gamma\in\Gamma} \varphi \Rightarrow \bigvee_{\delta\in\Delta} \delta$ est une tautologie.

2.6.2 La méthode de la coupure

Introduite en 1965 par Robinson, la résolution est un système formel pour le calcul des prédicats qui utilise des formules sous formes de clauses ; nous présenterons ce système plus tard (Section 3.8). Ici, nous allons présenter la méthode de la coupure, qui n'est rien d'autre que la restriction de la résolution à la logique propositionnelle.

2.6.2.1 Le système

Ce système formel dérive des nouvelles clauses à partir de clauses données. Il n'a pas d'axiomes, et comporte deux règles d'inférence (voir aussi la Figure 2.4) :

Factorisation : si une clause contient deux fois le même littéral, on en supprime une copie : on infère la clause $C \vee \ell$ à partir de la clause $C \vee \ell \vee \ell$.

Coupure (ou règle de résolution) : si deux clauses contiennent l'une un symbole propositionnel et l'autre sa négation ($C \vee p$ et $C' \vee \neg p$), on infère la clause $C \vee C'$, appelée **résultante** ou **résolvante** de $C \vee p$ et $C' \vee \neg p$ (on utilisera la notation $\rho(p, C \vee p, C \vee \neg p)$ pour dénoter cette clause).

$\dfrac{C \vee \ell \vee \ell}{C \vee \ell}$ factorisation	
$\dfrac{C \vee \ell \qquad C' \vee \neg\ell}{C \vee C'}$ coupure	$\dfrac{\ell \qquad \neg\ell}{\bot}$ coupure (cas particulier)

FIGURE 2.4 – Règles d'inférence pour la méthode de la coupure

On peut utiliser ce système formel pour :

1. montrer qu'une formule est contradictoire (ou admet un modelé) ;
2. montrer qu'une formule est une tautologie (ou non) : on montre que sa négation est contradictoire. On dit que la méthode de la coupure est *réfutationnelle* : pour prouver la formuel φ, on suppose $\neg\varphi$ et on montre que cela conduit à une contradiction.

Exemple 2.72. Soit la formule $\varphi = (p \lor r \lor s) \land (r \lor \neg s) \land \neg r \land \neg p$; cette formule s'écrit comme l'ensemble de clauses $\mathcal{C} = \{(p \lor r \lor s),\ (r \lor \neg s),\ \neg r,\ \neg p\}$. Voici une dérivation par résolution à partir de cet ensemble de clauses :

$$\dfrac{\dfrac{\dfrac{\dfrac{p \lor r \lor s \qquad r \lor \neg s}{p \lor r \lor r}\,\text{coupure}}{p \lor r}\,\text{factorisation} \qquad \neg r}{p}\,\text{coupure} \qquad \neg p}{\bot}\,\text{coupure}$$

Puisqu'on arrive à dériver la clause vide, la formule φ est insatisfiable.

Exemple 2.73. L'exemple suivant montre qu'on peut avoir besoin d'utiliser deux fois une clause pour dériver la clause vide. Soit φ la formule $((p \Rightarrow q) \land (q \Rightarrow p) \land (p \lor q)) \Rightarrow (p \land q)$. On veut montrer que φ est est une tautologie, on met donc $\neg\varphi$ sous forme clausale, on obtient l'ensemble de clause $\mathcal{C} = \{(p \lor \neg q), (\neg p \lor q), (p \lor q), (\neg p \lor \neg q)\}$. Une preuve par la méthode de la coupure est la suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p \lor \neg q \qquad \neg q \lor \neg p}{\neg q \lor \neg q}\,\text{coupure}}{\neg q}\,\text{factorisation} \qquad \neg p \lor q}{\neg p}\,\text{coupure} \qquad p \lor q}{q}\,\text{coupure} \qquad \neg q}{\bot}\,\text{coupure}$$

Donc nous avons montré qu'on peut deriver $\bot$ à partir de la clause vide, ce qu'implique que $\neg\varphi$ est insatisfaisable, et donc φ est une tautologie.

Remarquez que la clause $\neg q$ est utilisée deux fois ; aussi, l'exemple montre que la règle de factorisation est nécessaire, sans elle toutes les résolvantes ont deux littéraux et on ne peux donc jamais dériver la clause vide.

2.6.2.2 Correction et complétude

Nous allons écrire $\mathcal{C} \vdash_R \psi$ si nous pouvons dériver la clause ψ à partir des clauses de l'ensemble $\mathcal{C}$, par application successive des règles de la méthode de la coupure.

Théorème 2.74. *La résolution est correcte ; c'est-à-dire, si $\mathcal{C} \vdash_R \bot$, alors $\mathcal{C}$ est insatisfaisable.*

Démonstration. $\mathcal{C} \vdash_R \bot$ signifie qu'il existe une suite finie d'ensemble de clauses $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$ vérifiant :

— $\mathcal{C}_0 = \mathcal{C}$;

— pour tout $i \in [0, n-1]$, on obtient $\mathcal{C}_{i+1}$ en appliquant une des deux règles de la méthode de la copure à $\mathcal{C}_i$;

— $\mathcal{C}_n$ contient la clause $\bot$.

On remarque facilement qu'alors pour tout $i \in [0, n-1]$, $\mathcal{C}_{i+1}$ a exactement les mêmes modèles que $\mathcal{C}_i$. En effet,

— si la règle appliquée est la coupure, alors il existe une variable p et deux clauses C_1, C_2 dans $\mathcal{C}_i$ telles que $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{R(p, C_1, C_2)\}$, où $R(p, C_1, C_2)$ est la résolvante de C_1 et C_2 par rapport à p. Trivialement, $R(p, C_1, C_2)$ est une conséquence logique de $\{C_1, C_2\}$ et donc une conséquence logique de $\mathcal{C}_i$. La Proposition 2.34 permet donc de conclure immédiatement que $\mathcal{C}_{i+1}$ et $\mathcal{C}_i$ ont exactement les mêmes modèles.

— si la règle appliquée est la factorisation, il est évident que la modèles de $\mathcal{C}_{i+1}$ et $\mathcal{C}_i$ sont les mêmes.

En conclusion, $\mathcal{C}_0$ et $\mathcal{C}_n$ ont exactement les mêmes modèles, $\mathcal{C}_n$ contient la clause $\bot$, il est donc est insatisfaisable, donc $\mathcal{C} = \mathcal{C}_0$ est insatisfaisable. □

Définition 2.75. Un ensemble de clauses (factorisées) est dit *saturé* si on ne peut pas produire des nouvelles clauses par application de la règle de coupure.

Théorème 2.76. *La résolution est complète. C'est-à-dire : si $\mathcal{C}$ est insatisfaisable, alors $\mathcal{C} \vdash_R \bot$.*

Démonstration. Par le théorème de compacité, nous pouvons assumer que $\mathcal{C}$ est un ensemble fini ; donc les symboles propositionnels appairassant dans ses clauses sont en nombre fini.

Nous allons montrer que si $\mathcal{C} \not\vdash_R \bot$, alors $\mathcal{C}$ est satisfaisable. Or la condition $\mathcal{C} \not\vdash_R \bot$ revient à dire qu'à partir de $\mathcal{C}$, on peut engendrer, par application itérée de toutes les possibles règles, un ensemble $\mathcal{S}$ saturé (et possiblement infini) tel que $\bot \notin \mathcal{S}$ et tel que l'énsemble des ses variables propositionnelles est aussi fini. En fait nous allons montrer que un tel ensemble $\mathcal{S}$ est est satisfaisable ; le résultat découle ensuite du fait que $\mathcal{C} \subseteq \mathcal{S}$ et donc $\mathsf{mod}(\mathcal{S}) \subseteq \mathsf{mod}(\mathcal{C})$.

On suppose donc que $\mathcal{S}$ est saturé et ne contient pas la clause $\bot$ et on montre que $\mathcal{S}$ a un modèle.

On fait la preuve par récurrence sur le nombre n de symboles propositionnels de $\mathcal{S}$.

Cas de base : $n = 1$. Car il est saturé, $\mathcal{S}$ ne peut contenir à la fois les deux clauses p et $\neg p$, sinon il contiendrait aussi donc la clause vide. Donc $\mathcal{S}$ admet un modèle.

Pas d'induction. Supposons la propriété vraie pour n et que $\mathcal{S}$ contient $n+1$ symboles propositionnels. Soit p un symbole propositionnel apparaissant dans une clause de $\mathcal{S}$. On définit :

$$\mathcal{S}_0 = \{\, C \in \mathcal{S} \mid \neg p \notin C \,\}, \qquad \mathcal{S}'_0 = \mathcal{S}_0[p \leftarrow \bot],$$
$$\mathcal{S}_1 = \{\, C \in \mathcal{S} \mid p \notin C \,\}, \qquad \mathcal{S}'_1 = \mathcal{S}_1[p \leftarrow \top].$$

Remarquez que $\bot \in \mathcal{S}'_0$ implique que $p \in \mathcal{S}_0$; aussi $\mathcal{S}'_0$ est saturé car, pour $q \neq p$, on voit bien que

$$R(q, C_1[p \leftarrow \bot], C_2[p \leftarrow \bot]) = R(q, C_1, C_2)[p \leftarrow \bot].$$

De même, $\mathcal{S}'_1$ est saturé, et $\bot \in \mathcal{S}'_1$ implique $\neg p \in \mathcal{S}_1$. En particulier, si $\bot \in \mathcal{S}'_0 \cap \mathcal{S}'_1$, alors $p \in \mathcal{S}_0 \subseteq \mathcal{S}$ et $\neg p \in \mathcal{S}_1 \subseteq \mathcal{S}$; car $\mathcal{S}$ est saturé, alors on obtient $\bot \in \mathcal{S}$, une contradiction. Un des $\mathcal{S}'_i$ est non vide et ne contient pas la clause vide ; sans perte de généralité, nous pouvons supposer qu'il s'agit de $\mathcal{S}'_1$. Par hypothèse d'induction, il y a un modèle v sur $\textsc{Prop} - \{p\}$. Alors l'extension du modèle de $\mathcal{S}'_1$ en posant $v(p) = 1$ est un modèle de $\mathcal{S}$. □

2.6.2.3 L'algorithme de résolution

On peut essayer d'extraire, de le méthode de la coupure, un algorithme pour décider si une formule φ est une tautologie : pour prouver qu'une formule φ est une tautologie, on peut procéder en trois étapes :

1. mettre $\neg\varphi$ sous forme clausale, $\neg\varphi = C_1 \wedge \cdots \wedge C_n$, où toute claiuse est factorisée ;
2. remplacer la conjonction de clauses $C_1 \wedge \ldots \wedge C_n$ par l'ensemble $\mathcal{C} = \{C_1, \ldots, C_n\}$;
3. saturer l'ensemble en rajoutant à $\mathcal{C}$ les clauses (factorisés) qu'on peut déduire à partir des deux règles de résolution.

Exemple 2.77. Si on considère l'ensemble de clauses $\mathcal{C} = \{p \vee \neg q, q \vee \neg p\}$. La méthode engendre $\{p \vee \neg p, q \vee \neg q, p \vee \neg q, q \vee \neg p\}$ (et rien de plus, mais il faut un raisonnement pour le prouver !) qui a un modèle $v(p) = 1$, $v(q) = 1$. Donc la formule $\neg((p \vee \neg q) \wedge (q \vee \neg p))$ n'est pas une tautologie.

L'algorithme termine lorsque :
— soit l'application des règles ne modifie plus l'ensemble de clauses ;
— soit on vient d'ajouter la clause $\bot$ et donc la formule initiale est insatisfaisable.

La méthode précédente n'a pas un vrai critère de terminaison autre que l'ensemble des clauses contient $\bot$. Si la formule à démontrer n'est pas un théorème, il faut soit montrer à la main qu'il n'est pas possible d'engendrer des résolvantes qui ne sont pas déjà dans l'ensemble de clauses, soit avoir un critère permettant d'arrêter de générer de nouvelles résolvantes (souvent en utilisant des stratégies de résolution). A la main, sur des exemples simples on peut souvent arrêter la résolution car on obtient un ensemble de clauses (clauses initiales plus toutes les clauses déduites) pour lequel on trouve un modèle. Ce modèle est donc un contre-exemple à la formule initiale qui n'est donc pas prouvable (c'est une conséquence des deux propriétés de complétude et correction qui suivent).

On voit sur l'exemple suivant que la résolution peut générer un ensemble infini de clauses et donc ne pas terminer.

Exemple 2.78. $\mathcal{C} = \{p \vee \neg p \vee \neg q\}$. On commence par appliquer la règle de résolution, et on obtient la clause $p \vee \neg p \vee \neg q \vee \neg q$, si lui applique la règle de factorisation, on retombe sur la clause initiale $p \vee \neg p \vee \neg q$, ce qui est inutile. On ne peut donc que réappliquer la résolution aux clauses $p \vee \neg p \vee \neg q$ et $p \vee \neg p \vee \neg q \vee \neg q$.

$$\cfrac{\cfrac{\cfrac{p \vee \neg p \vee \neg q \qquad p \vee \neg p \vee \neg q}{p \vee \neg p \vee \neg q \vee \neg q}\text{coupure} \qquad p \vee \neg p \vee \neg q}{p \vee \neg p \vee \neg q \vee \neg q \vee \neg q}\text{coupure} \qquad p \vee \neg p \vee \neg q}{\vdots}\text{coupure}$$

La méthode va générer les clauses de la forme $p \vee \neg p \vee \neg q \vee \neg q \vee \ldots \neg q$ et aucune autre (la encore un raisonnement est à faire). Donc la méthode ne termine pas et engendre un ensemble de clauses infini qui ne contient pas la clause vide. On peut conclure soit en trouvant directement un modèle (on peut en trouver plusieurs ici) soit en appliquant un résultat qui va être démontré dans la preuve de complétude : un ensemble de clauses saturé par résolution qui ne contient pas la clause vide a un modèle. Donc la formule $p \vee \neg p \vee \neg q$ n'est pas une tautologie.

On peut modifier la méthode pour obtenir un algorithme qui se termine toujours, si on se restreigne à considérer des clauses qui sont factorisées.

Définition 2.79. Un clause C est *factorisée* si elle ne contient pas deux occurrences du même littéral.

Si C est une clause, nous allons dénoter par $F(C)$ la clause factorisée obtenue de C par une suite de règles de factorisation.

Algorithme Res
entrée : un ensemble de clauses $\mathcal{C}$ factorisés sortie : $insatisfaisable$ si $\mathcal{C} \models \bot$ ou $satisfaisable$ sinon
si $\bot \in \mathcal{C}$, alors retourner : *insaitisfaisable* ; (réduire l'ensemble $\mathcal{C}$ via la règle de subsumption ;) si $C \vee p, C' \vee \neg p \in \mathcal{C}$, avec $F(R(p, C \vee p, C' \vee \neg p)) \notin \mathcal{C}$, alors retourner $Res(\mathcal{C} \cup \{ F(R(p, C \vee p, C' \vee \neg p)) \})$; sinon retourner : *satisfaisable*.

FIGURE 2.5 – L'algorithme de coupure (ou de résolution)

La terminaison de l'algorithme est maintenant assurée par le fait que le nombre de clause factorisées ayant au plus n symboles propositionnels est borné supérieurement par une fonction de n.

Exercice 2.80. Estimer cette borne supérieure.

Rappelons qu'une clause C_0 subsume une clause C_1 si $C_0 \subseteq C_1$ (i.e., si $C_1 \equiv C_0 \vee C_1'$, ou encore tout littéral apparaissant dans C_0 apparaît aussi dans C_1) ; si C_0 subsume C_1, alors $C_0 \models C_1$, et donc si $C_0, C_1 \in \mathcal{C}$, alors $\mathsf{mod}(\mathcal{C}) = \mathsf{mod}(\mathcal{C} \setminus \{ C_1 \})$. L'algorithme *Res* peut donc être optimisé si on mantient un ensemble de clauses tel que, chaque fois que $C_0, C_1 \in \mathcal{C}$, alors ni C_0 subsume C_1,, ni C_1 subsume C_0.

2.6.3 Systèmes de preuve à la Hilbert

Bien que la résolution soit un calcul très adapté à l'implantantion sur ordinateur, c'est assez difficile de reconnaître dans ce calcul le raisonnement courant, mathématique ou non. Considérez, par exemple, l'inférence mathématique suivante :

« Si un graphe est complet, alors il possède une seule clique maximale. $\mathcal{K}_4$ est un graphe complet. Par conséquent, $\mathcal{K}_4$ possède une seule clique maximale. »

Nous reconnaissons ici un schéma d'inférence bien connu, appellé *modus ponens*. Il est de la forme forme suivante [1] :

$$\frac{X \Rightarrow Y \qquad X}{Y} \tag{MP}$$

C'est-à-dire : *si X implique Y (est vraie), et X (est vraie), alors Y (est vraie).* Le calcul de la résolution, en traitant des formules clausales seulement, ne permet pas de comprendre quelles sont les inférences correctes qui règlent le connecteur logique $\Rightarrow$.

Nous allons donc présenter un deuxième système formel pour la logique propositionnelle, qui étudie et simule le raisonnement mathématique de près, en donnant à l'implication son rôle traditionnel de premier plan. Rappelez vous que toute formule est équivalent à (donc peut se coder dans) une formule construite à partir de l'implication et de la négation. On peut donc supposer que les connecteurs logique sont ces deux seulement.

Dans ses démonstrations un mathématicien utilise des axiomes et des règles d'inférence pour démontrer des propositions. Nous allons donc spécifier quels sont les axiomes, et quelles sont les règles d'inférence.

Axiomes. Sont les suivants :

$$X \Rightarrow (Y \Rightarrow X) \tag{A1}$$

$$(X \Rightarrow (Y \Rightarrow Z)) \Rightarrow ((X \Rightarrow Y) \Rightarrow (X \Rightarrow Z)) \tag{A2}$$

$$(\neg X \Rightarrow Y) \Rightarrow ((\neg X \Rightarrow \neg Y) \Rightarrow X)\,, \tag{A3}$$

où X, Y, Z sont des formules quelconques [2].

Règles d'inférence. Une seule, le modus ponens (MP), ou X, Y sont des formules quelconques.

Définition 2.81. Soit $\Gamma \subseteq \mathcal{F}_{\text{cp}}$ et $\varphi \in \mathcal{F}_{\text{cp}}$. Une démonstration (ou preuve) de φ à partir des hypothèses Γ est une liste ordonnée $\varphi_1, \ldots, \varphi_n$ telle que

- $\varphi_i \in \mathcal{F}_{\text{cp}}$ for $i = 1, \ldots, n$,
- $\varphi_n = \varphi$,
- pour tout $i = 1, \ldots, n$, ou bien
  - φ_i est un axiome, ou bien
  - $\varphi_i \in \Gamma$, ou bien
  - il existe $j, k < i$ tels que φ_i peut s'inférer de φ_j et φ_k via la règle du modus-ponens.

Nous allons écrire $\Gamma \vdash \varphi$ s'il existe une démonstration Π de φ à partir des hypothèses Γ. Nous allons écrire $\vdash \varphi$ si $\varnothing \vdash \varphi$ et dire alors que φ *est un théorème.*

Exemple 2.82. Soit $\varphi, \psi \in \mathcal{F}_{\text{cp}}$. Nous avons $\vdash \varphi \Rightarrow (\psi \Rightarrow \varphi)$ car la liste de longuer 1 composée par $\varphi \Rightarrow (\psi \Rightarrow \varphi)$ est une preuve de cette formule.

Exemple 2.83. Considérons la suite des formules suivantes :

1,	$(\psi \Rightarrow ((\psi \Rightarrow \psi) \Rightarrow \psi)) \Rightarrow ((\psi \Rightarrow (\psi \Rightarrow \psi)) \Rightarrow (\psi \Rightarrow \psi))$	A2, avec ψ, $(\psi \Rightarrow \psi)$ et ψ
2,	$\psi \Rightarrow ((\psi \Rightarrow \psi), \Rightarrow \psi)$	A1, avec ψ et $(\psi \Rightarrow \psi)$,
3,	$(\psi \Rightarrow (\psi \Rightarrow \psi)) \Rightarrow (\psi \Rightarrow \psi)$	MP, de 1 et 2
4,	$\psi \Rightarrow (\psi \Rightarrow \psi)$	A1, avec ψ et ψ
5,	$\psi \Rightarrow \psi$	MP, de 3 et 4

1. La structure de l'inférence exemplifiée est en fait bien plus complexe ; cet exemple peux pleinement se comprendre dans le cadre de la logique du premier ordre.
2. On devrait donc parler de *schémas d'axiomes* au lieu d'*axiome*.

Cette suite est une preuve de $\psi \Rightarrow \psi$ à partir de l'ensemble vide, et donc nous allons écrire $\vdash \psi \Rightarrow \psi$. Remarquons que ψ est une formule arbitraire.

Si $\Gamma \vdash \varphi$, nous pensons au couple (Γ, φ) comme une *règle d'inférence dérivée* (le mot technique étant *admissible*).

Exemple 2.84. Clairement, nous avons $\{ X, X \Rightarrow Y \} \vdash Y$.

Exemple 2.85. Nous avons $\{ X \Rightarrow Y, Y \Rightarrow Z \} \vdash X \Rightarrow Z$, pour tout $X, Y, Z \in \mathcal{F}_{\text{cp}}$. Voici la (quasi)-démonstration qui en est témoin :

1,	$Y \Rightarrow Z$	$Y \Rightarrow Z \in \Gamma$
2,	$(Y \Rightarrow Z) \Rightarrow (X \Rightarrow (Y \Rightarrow Z))$	A1, avec $(Y \Rightarrow Z)$ et X
3,	$X \Rightarrow (Y \Rightarrow Z)$	MP, de 1 et 2
4,	$(X \Rightarrow (Y \Rightarrow Z)) \Rightarrow ((X \Rightarrow Y) \Rightarrow (X \Rightarrow Z))$	A2, avec X, Y et Z
5,	$(X \Rightarrow Y) \Rightarrow (X \Rightarrow Z)$	MP, de 3 et 4
6,	$X \Rightarrow Y$	$X \Rightarrow Y \in \Gamma$
7,	$X \Rightarrow Z$	MP, de 5 et 6

Nous pouvons écrire cette règle d'inférence admissible, que nous avons trouvé, d'une facon semblable au modus ponens :

$$\frac{X \Rightarrow Y \qquad Y \Rightarrow Z}{X \Rightarrow Z} \tag{MB}$$

Cette règle d'inférences est connue, parmi les syllogismes, comme « Modus Barbara ».

Les exemples précédents montrent qu'il peut se révéler assez difficile construire des preuves qui soient témoins qu'une formule, bien que simple, est un théorème. Nous allons donc élargir la notion de preuve, afin de rendre la construction des démonstrations une tache moins complexe.

Lemme 2.86. *Soit* $\Pi = \varphi_1, \ldots, \varphi_n$ *une suite de formules telles que, pour tout* $i = 1, \ldots, n$, *on a un de ces cas :*
- — φ_i *est un axiome,*
- — $\varphi_i \in \Gamma$,
- — φ_i *est inférée à partir des formules* φ_i, φ_k, *avec* $j, k < i$, *via la règle MP,*
- — $\Gamma \vdash \varphi_i$,
- — $\Gamma \cup \Sigma \vdash \varphi_i$, *où* Σ *est un sous-ensemble de* $\{ \varphi_1, \ldots, \varphi_{i-1} \}$.

Alors $\Gamma \vdash \varphi_n$.

Observez que le lemme précédent s'énonce concisement de la façon suivante : *soit* $\Pi = \varphi_1, \ldots, \varphi_n$ *une suite de formules telles que, pour tout* $i = 1, \ldots, n$, $\Gamma \cup \{ \varphi_1, \ldots, \varphi_{i-1} \} \vdash \varphi_i$ *; alors* $\Gamma \vdash \varphi_n$.

Démonstration. Pour tout $i = 1, \ldots, n$, soit $\varphi_{i,1} \ldots, \varphi_{i,n_i} = \varphi_i$ une démonstration de φ_i à partir de $\Gamma \cup \{ \varphi_1, \ldots, \varphi_{i-1} \} \vdash \varphi_i$. La suite obtenue de Π en remplaçant, pour chaque $i = 1, \ldots, n$, φ_i par la suite otenue de $\varphi_{i,1} \ldots, \varphi_{i,n_i}$ en effaçant les formules dans $\{ \varphi_1, \ldots, \varphi_{i-1} \}$, est une démonstration de φ_n à partir de Γ. □

Exemple 2.87. Comme exemple du Lemme 2.86, nous allons montrer que la lois d'involution

$$\neg\neg\psi \Rightarrow \psi$$

est un théorème. Voici une (quasi)-démonstration :

1,	$\neg\psi \Rightarrow \neg\psi$	car $\vdash \neg\psi \Rightarrow \neg\psi$, voir 2.83
2,	$(\neg\psi \Rightarrow \neg\psi) \Rightarrow ((\neg\psi \Rightarrow \neg\neg\psi) \Rightarrow \psi)$	A3, avec ψ et $\neg\psi$
3,	$(\neg\psi \Rightarrow \neg\neg\psi) \Rightarrow \psi$	MP, de 1 et 2
4,	$\neg\neg\psi \Rightarrow (\neg\psi \Rightarrow \neg\neg\psi)$	A1, avec $\neg\neg\psi$ et $\neg\psi$
5,	$\neg\neg\psi \Rightarrow \psi$	MB, de 4 et 3

Le théorème de Déduction

Théorème 2.88 (de Déduction)**.** *On a*

$$\Gamma \cup \{ \varphi \} \vdash \psi \text{ ssi } \Gamma \vdash \varphi \Rightarrow \psi \,.$$

Démonstration. Supposons d'abord que Π est une preuve de $\Gamma \vdash \varphi \Rightarrow \psi$; la concaténation de Π avec le morceau de preuve suivant :

1,	$\varphi \Rightarrow \psi$	$\dots$
2,	φ	$\varphi \in \Gamma$
3,	ψ	MP, de 1 et 2

est alors une preuve Π' de $\Gamma \cup \{ \varphi \} \vdash \psi$.

Nous allons prouver l'implication inverse—si $\Gamma \cup \{ \varphi \} \vdash \psi$ alors $\Gamma \vdash \varphi \Rightarrow \psi$—par induction sur la longueur k d'une preuve Π de $\Gamma \cup \{ \varphi \} \vdash \psi$. Ecrivons donc $\Gamma \cup \{ \varphi \} \vdash_k \varphi$ pour dire qu'il existe une preuve Π de ψ à partir de $\Gamma \cup \{ \varphi \}$ de longuer au plus k.

Si $k = 1$, alors on a un de ces trois cas : (1) ψ est un axiome, (2) $\psi \in \Gamma$, (3) $\psi = \varphi$. Dans le premier deux cas ((1) ou (2)) nous avons que

1,	ψ	ψ est un axiome, ou $\psi \in \Gamma$
2,	$\psi \Rightarrow (\varphi \Rightarrow \psi)$	A1, avec ψ et φ
3,	$\varphi \Rightarrow \psi$	MP, de 1 et 2

est une preuve Π' de $\Gamma \vdash \varphi \Rightarrow \psi$. Sinon (cas (3)) on a $\varphi = \psi$; nous savons que $\vdash \psi \Rightarrow \psi$ (voir Exemple **??**), donc $\Gamma \vdash \psi \Rightarrow \psi \ \ (= \varphi \Rightarrow \psi)$.

Supposons maintenant que $\Gamma \vdash_{k+1} \varphi$ et soit Π une preuve de longueur au plus $k+1$; par hypothese d'induction, si $\Gamma \cup \{ \varphi \} \vdash_k \psi$, alors $\Gamma \vdash \varphi \Rightarrow \psi$.

Considérons comment ψ a été justifiée. Si ψ est un axiome, appartient à Γ ou encore $\psi = \varphi$, nous pouvons construire une preuve de $\Gamma \vdash \varphi \Rightarrow \psi$ exactement comme auparavant (le cas $k = 1$).

Sinon, ψ a été inférée via la règle du modus ponens, à partir de deux formules ψ_1 et $\psi_1 \Rightarrow \psi_2$ qui précèdent ψ dans cette preuve. Nous avons alors que $\Gamma \cup \{ \varphi \} \vdash_k \psi_1$ et $\Gamma \cup \{ \varphi \} \vdash_k \psi_1 \Rightarrow \psi_2$. Par hypothèse d'induction, nous disposons d'une preuve Π_1 témoin de $\Gamma \vdash \varphi \Rightarrow \psi_1$ et d'une preuve Π_2 témoin de $\Gamma \vdash \varphi \Rightarrow (\psi_1 \Rightarrow \psi_2)$. Nous pouvons concatener les preuves Π_1 et Π_2 avec le morceau de preuve suivant

1,	$(\varphi \Rightarrow (\psi_1 \Rightarrow \psi_2)) \Rightarrow ((\varphi \Rightarrow \psi_1) \Rightarrow (\varphi \Rightarrow \psi_2))$	A2, avec φ, ψ_1 et ψ_2
2,	$(\varphi \Rightarrow \psi_1) \Rightarrow (\varphi \Rightarrow \psi_2)$	MP, de $\varphi \Rightarrow (\psi_1 \Rightarrow \psi_2)$ et 1
3,	$\varphi \Rightarrow \psi_2$	MP, de $\varphi \Rightarrow \psi_1$ et 2

pour obtenir une preuve Π' témoin de $\Gamma \vdash \varphi \Rightarrow \psi_2$. □

TODO : ajouter discussion de ce théorème

Le théorème de Correction et Complétude

Lemme 2.89. *Les axiomes A1, A2, et A3, sont des tautologies, pour tout instanciation de X, Y, Z par des formules.*

Proposition 2.90 (Correction)**.** *Si $\Gamma \vdash \varphi$, alors $\Gamma \models \varphi$.*

Démonstration. Soit $\varphi_1, \dots, \varphi_k$ une preuve de φ à partir de Γ, et soit $v \in \mathsf{mod}(\Gamma)$. Nous allons montrer que $v(\varphi_i) = 1$ pour tout $i = 1, \dots, k$.

Pour $i = 1$, alors $\varphi_i \in \Gamma$, ou bien φ_i est un axiome. Si $\varphi_i \in \Gamma$, alors $v(\varphi_i) = 1$ car $v \in \mathsf{mod}(\Gamma)$, et si φ_i est un axiome, alors $v(\varphi_i) = 1$ par le Lemme 2.89.

Supposons donc que $i > 1$ et que $v(\varphi_j) = 1$ pour $j < i$ (l'hypothèse d'induction). Encore une fois, si φ_i est un axiome ou appartient à Γ, alors pouvons inférer que $v(\varphi_i) = 1$ comme auparavant.

Sinon, φ_i est inférée via la règle MP à partir de formules φ_j et $\varphi_j \Rightarrow \varphi_i$ telles que (par hypothèse d'induction) $v(\varphi_j) = 1$ et $v(\varphi_j \Rightarrow \varphi_i) = 1$. En étudiant la table de vérité du connecteur logique $\Rightarrow$, on s'aperçoit alors que $v(\varphi_i) = 1$. □

Avant prouver la complétude du système de preuves, montrons quelques résultats qui sera utile pendant la démonstration. La preuve des ces résultats sera accomplie en TD.

Lemme 2.91. *Pour toutes formules $\varphi, \psi \in \mathcal{F}_{cp}$, nous avons :*

1. $\varphi, \neg\varphi \vdash \psi$;
2. $\neg(\varphi \Rightarrow \psi) \vdash \varphi$;
3. $\neg(\varphi \Rightarrow \psi) \vdash \neg\psi$;
4. $\Gamma \cup \{\varphi\} \vdash \psi$ *et* $\Gamma \cup \{\neg\varphi\} \vdash \psi$ *implique* $\Gamma \vdash \psi$.
5. $\Gamma \cup \{\varphi\} \vdash \psi$ *et* $\Sigma \vdash \varphi$ *implique* $\Gamma \cup \Sigma \vdash \psi$.

Ces résultats peuvent se comprendre comme suit. (1) établie que n'importe quelle formule ψ est démontrable à partir d'hypothèses contradictoires. Pour (2) et (3), rappelons que la conjonction se code (dans la langage dont les seuls connecteurs logiques sont $\Rightarrow$ et $\neg$) par :

$$\varphi \wedge \psi := \neg(\varphi \Rightarrow \neg\psi) .$$

Une conséquence immédiate de (2) et (3) et qu'on peut prouver φ et ψ à partir de Γ si $\varphi \wedge \psi \in \Gamma$, c'est-à-dire $\{\varphi \wedge \psi\} \vdash \varphi$ et $\{\varphi \wedge \psi\} \vdash \psi$.

Proposition 2.92 (Complétude). *Si $\Gamma \models \varphi$, alors $\Gamma \vdash \varphi$.*

Démonstration. Nous allons prouver que $\Gamma \nvdash \varphi$ implique que $\Gamma \not\models \varphi$.

Soit $\varphi_1, \ldots, \varphi_n, \ldots$ une enumeration de toutes les formules de $\mathcal{F}_{\text{cp}}$. Posons $\Gamma_0 = \Gamma$, et

$$\Gamma_{n+1} = \begin{cases} \Gamma_n \cup \{\varphi_{n+1}\} , & \text{si } \Gamma_n \cup \{\varphi_{n+1}\} \nvdash \varphi , \\ \Gamma_n , & \text{sinon} . \end{cases}$$

Posons enfin

$$\Gamma_\omega = \bigcup_{n \geq 0} \Gamma_n .$$

Remarquons que $\Gamma_n \nvdash \varphi$, pour tout $n \geq 0$, et aussi les propriétés suivantes de Γ_ω :

1. $\Gamma_\omega \nvdash \varphi$. Car sinon, il existe un sous-ensemble fini $\Gamma' \subseteq \Gamma_\omega$ tel que $\Gamma' \vdash \varphi$, et ainsi il existe $n \geq 0$ avec $\Gamma' \subseteq \Gamma_n$; cela entraîne que $\Gamma_n \vdash \varphi$, ce qui contredit la définition de Γ_n.
2. $\Gamma_\omega \cup \{\psi\} \vdash \varphi$, pour toute formule $\psi \notin \Gamma_\omega$. En fait supposons, que $\psi \notin \Gamma_\omega$ et $\psi = \varphi_{n+1}$. Or car $\psi \notin \Gamma_\omega$, cela implique que $\Gamma_n = \Gamma_{n+1}$, et la seule raison de cette identité est que $\Gamma_n \cup \{\psi\} \vdash \varphi$. A fortiori, $\Gamma_\omega \cup \{\psi\} \vdash \varphi$.
3. $\psi \in \Gamma_\omega$ ssi $\neg\psi \notin \Gamma_\omega$. En fait, si $\psi, \neg\psi \in \Gamma_\omega$, alors $\Gamma_\omega \vdash \varphi$, ce qui contredit (1) ; si par contre $\psi, \neg\psi \notin \Gamma_\omega$, alors $\Gamma_\omega \cup \{\psi_n\} \vdash \varphi$ et $\Gamma_\omega \cup \{\neg\psi_n\} \vdash \varphi$; par le Lemme 2.91, nous avons encore $\Gamma_\omega \vdash \varphi$.
4. $\Gamma_\omega \vdash \psi$ implique $\psi \in \Gamma_\omega$. Si $\psi \notin \Gamma_\omega$, alors $\Gamma_\omega \cup \{\psi\} \vdash \varphi$. Par le Lemme 2.91, on a alors $\Gamma_\omega \vdash \varphi$.

Soit v la valuation telle que $v(p) = 1$ ssi $p \in \Gamma_\omega$. Montrons, par induction sur les formules, que la proprietée suivante :

— $\psi \in \Gamma_\omega$ implique $v(\psi) = 1$;
— $\psi \notin \Gamma_\omega$ implique $v(\psi) = 0$;

est vraie de toute formule $\psi \in \mathcal{F}_{\text{cp}}$.

Si $\psi = p \in \textsc{Prop}$ est une formule atomique, alors cela est vrai pour définition de v.

Supposons $\psi = \psi_1 \Rightarrow \psi_2 \in \Gamma_\omega$. Nous devons montrer que $v(\psi_1 \Rightarrow \psi_2) = 1$, ce qui revient à dire que $v(\psi_1) = 1$ implique $v(\psi_2) = 1$. Supposons donc que $v(\psi_1) = 1$; par hypothese d'induction, nous avons $\psi_1 \in \Gamma_\omega$. Car aussi $\psi_1 \Rightarrow \psi_2 \in \Gamma_\omega$ et Γ_ω est fermee par rapport au MP (à cause du remarque 4.), nous avons $\psi_2 \in \Gamma_\omega$, et par HI, $v(\psi_2) = 1$.

Supposons maintenant que $\psi = \psi_1 \Rightarrow \psi_2 \notin \Gamma_\omega$. Nous devons montrer que $v(\psi_1 \Rightarrow \psi_2) = 0$, ce qui revient à dire que $v(\psi_1) = 1$ et $v(\psi_2) = 0$, ou, par HI, $\psi_1 \in \Gamma_\omega$ et $\psi_2 \notin \Gamma_\omega$. Nous avons $\neg(\psi_1 \Rightarrow \psi_2) \in \Gamma$, $\neg(\psi_1 \Rightarrow \psi_2) \vdash \psi_1$, $\neg(\psi_1 \Rightarrow \psi_2) \vdash \neg\psi_2$, et donc (avec les remarques 3. et 4.) $\psi_1 \in \Gamma_\omega$ et $\psi_2 \notin \Gamma_\omega$.

Pour finir, nous allons supposer que $\psi = \neg\psi_1$. Si $\psi \in \Gamma$, alors $\psi_1 \notin \Gamma$ et par HI $v(\psi_1) = 0$, donc $v(\psi) = 1$. Si $\psi \notin \Gamma$, alors $\psi_1 \in \Gamma$ et par HI $v(\psi_1) = 1$, donc $v(\psi) = 0$. □

Nous pouvons finalement énoncer le Théorème de Correction et Complétude :

Théorème 2.93. *La relation* $\Gamma \vdash \varphi$ *est vraie si, et seulement si, la relation* $\Gamma \models \varphi$ *est vraie.*

Veuillez remarquer la non-trivialité de ce Théorème, qui établie l'identité entre la notion de prouvabilité—définie via axiomes et règles d'inférence—et la notion de conséquence logique—définie via l'algèbre des sous-ensembles.

TODO : Ajouter discussion de ces Théorèmes

Les règles du calcul des séquents

Règles structurelles

$$\frac{\Gamma, \varphi, \varphi \vdash \Delta}{\Gamma, \varphi, \vdash \Delta} \ (G_{Contr}) \qquad \frac{\Gamma \vdash \Delta, \varphi, \varphi}{\Gamma \vdash \Delta, \varphi} \ (D_{Contr})$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, \varphi \vdash \Delta} \ (G_{Aff}) \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \varphi} \ (D_{Aff})$$

Règles logiques

$$\frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta} \ (G_{\neg}) \qquad \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta} \ (D_{\neg})$$

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta} \ (G_{\wedge}) \qquad \frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta} \ (D_{\wedge})$$

$$\frac{\Gamma, \varphi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta} \ (G_{\vee}) \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta} \ (D_{\vee})$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \Rightarrow \psi \vdash \Delta} \ (G_{\Rightarrow}) \qquad \frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \Rightarrow \psi, \Delta} \ (D_{\Rightarrow})$$

Règle de coupure

$$\frac{\Gamma_1 \vdash \varphi, \Delta_1 \quad \Gamma_2, \varphi \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} \ (C)$$

2.7 Résumé

Formules propositionnelles

Les briques de base des formules propositionnelles sont les **propositions** appelées aussi **symboles propositionnels** ou **atomes** ou **formules atomiques** ou **variables propositionnelles**. On note PROP l'ensemble des propositions. L'ensemble PROP_0 des **formules propositionnelles** est le plus petit ensemble contenant PROP et clos par l'application des connecteurs $\wedge, \vee, \neg, \Rightarrow$.

Un **littéral** est une formule atomique ou la négation d'une formule atomique. Une **clause disjonctive** est une disjonction de littéraux. Une **clause conjonctive** est une conjonction de littéraux. Une **formule conjonctive**, ou formule sous **forme normale conjonctive** (FNC), est une conjonction de clauses disjonctives. Une **formule disjonctive**, ou formule sous **forme normale disjonctive** (FND), est une disjonction de clauses conjonctives. En résumé, ces quatres notions recouvrent les formes suivantes :

$$\text{Clause conjonctive : } \bigwedge_{i=1}^{n} \ell_i \,; \qquad \text{Clause disjonctive : } \bigvee_{i=1}^{n} \ell_i \,;$$

$$\text{FNC : } \bigwedge_{j=1}^{m} \bigvee_{i=1}^{n} \ell_i^j \,; \qquad \text{FND : } \bigvee_{j=1}^{m} \bigwedge_{i=1}^{n} \ell_i^j .$$

où les ℓ_i et ℓ_i^j sont des littéraux.

Modèles d'une formule

Une **valuation** est une application de PROP dans $\{0, 1\}$. La notion de valuation peut-être étendue aux formules, ce qui permet de calculer la valeur d'une formule en fonction de la valeur de ses atomes. L'ensemble des valuation d'un ensemble de propositions PROP est noté $\mathsf{Val}(\text{PROP})$ (ou juste Val lorsqu'il n'y a pas d'ambiguité sur PROP). Un **modèle** d'une formule φ est une valuation v telle que $v(\varphi) = 1$. Si v est un modèle de φ on note $v \models \varphi$. On note $\mathsf{mod}(\varphi)$ l'ensemble des modèles de φ.

Une formule est **satisfaisable** si elle admet un modèle, **insatisfaisable** dans le cas contraire. Une **tautologie** (ou formule valide) est une formule vraie pour toute valuation. On note $\models \varphi$ pour dire que φ est une tautologie.

Une formule ψ est **conséquence logique** d'une formule φ si tout modèle de φ est un modèle de ψ. On note alors $\varphi \models \psi$. Deux formules φ et ψ sont dites **équivalentes** si $\mathsf{mod}(\varphi) = \mathsf{mod}(\psi)$, on note alors $\varphi \equiv \psi$.

Modèles d'un ensemble de formules

Un **modèle** d'un ensemble de formules Γ est une valuation v telle que $v(\varphi) = 1$ pour tout $\varphi \in \Gamma$. On note $\mathsf{mod}(\Gamma)$ l'ensemble des modèles de Γ. Un ensemble de formules Γ est **satisfaisable** ou **consistant** si il admet au moins un modèle, **contradictoire** (ou **insatisfaisable**) dans le cas contraire.

Une formule φ est **conséquence logique** d'un ensemble Γ de formules si et seulement si $\mathsf{mod}(\Gamma) \subseteq \mathsf{mod}(\varphi)$). On note alors $\Gamma \models \varphi$. On note $\text{cons}(\Gamma)$ l'ensemble des conséquences logiques de Γ. On a : (i) $\Gamma \models \varphi$ ssi $\Gamma \cup \{\neg\varphi\}$ est contradictoire. (ii) $\Sigma \subseteq \Gamma$ alors $\mathsf{mod}(\Gamma) \subseteq \mathsf{mod}(\Sigma)$.

Compacité : Un ensemble Γ de formules est consistant si et seulement si chaque partie finie de Γ est consistante. Un ensemble Γ de formules est inconsistant si et seulement il existe une partie finie de Γ consistante.

Système de preuve.

Une logique comporte une **syntaxe** (pour définir les formules), une **sémantique** (pour définir le sens d'une formule), un **système formel** (un calcul pour prouver une formule). Un système formel est **correct** s'il ne prouve pas de formules qui ne sont pas vraies ; il est **complet** s'il permet de prouver tout ce qui est vrai. La **résolution** et la **déduction naturelle** sont corrects et complets.

Bibliographie

[ES04] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer Berlin Heidelberg, 2004.