
Cours Logique et Calculabilité

L3 Informatique 2016/2017

Texte par Séverine Fratani et Luigi Santocanale
Version du 18 mars 2017

Table des matières

1	Calcul propositionnel	5
1.1	Introduction	5
1.2	Syntaxe du calcul propositionnel : les formules	5
1.3	Sémantique du calcul propositionnel	6
1.3.1	Modèles d'une formule	8
1.3.2	La conséquence logique (d'un ensemble de formules)	12
1.3.3	Décidabilité du calcul propositionnel	16
1.4	Equivalence entre formules	18
1.4.1	La substitution	18
1.4.2	Equivalences classiques	18
1.4.3	Formes normales	19
1.5	Le problème SAT	22
1.5.1	Définition du problème	22
1.5.2	Un problème NP-complet	22
1.5.3	Modélisation - Réduction à SAT	23
1.5.4	Algorithmes de résolution de SAT	26
1.5.5	Sous-classes de SAT	31
1.5.6	Les SAT-solvers	33
1.5.7	Applications	34
1.5.8	Sur la modélisation	36
1.6	Systèmes de preuves	37
1.6.1	La notion de système formel	37
1.6.2	La méthode de la coupure	37
1.7	Résumé	42
2	Calcul des prédicats	47
2.1	Introduction	47
2.2	Préliminaires - Exemple introductif	47
2.2.1	Produit d'un ensemble	48
2.2.2	Les relations	48
2.2.3	Les fonctions	48
2.2.4	Un exemple	48
2.3	Expressions et formules	50
2.3.1	Les termes	50
2.3.2	Le langage	51
2.3.3	Les formules du calcul des prédicats	51
2.3.4	Occurrences libres et liées d'une variable	52
2.4	Sémantique	53
2.4.1	Structures	54
2.4.2	Evaluation (des termes et) des formules	55
2.5	Manipulation de formules	60
2.5.1	Substitution de variables	60
2.5.2	Equivalences classiques	60
2.5.3	Formes Normales	62

2.6	Unification	65
2.6.1	Substitutions et MGUs	65
2.6.2	Algorithme d'unification	69
2.6.3	Correction et complétude	69
2.7	Résolution	72
2.7.1	Substitution, sur les formules propositionnelles	72
2.7.2	Les règles du calcul de la résolution	72
2.7.3	Correction du calcul de la résolution	73
2.7.4	Complétude du calcul de la résolution	74
2.7.5	Indécidabilité	76
2.7.6	Utilisation d'un démonstrateur automatique	77

Chapitre 1

Calcul propositionnel

1.1 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 « 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.

1.2 Syntaxe du calcul propositionnel : les formules

Le langage du calcul propositionnel est formé de :

- symboles propositionnels $\text{PROP} = \{p_1, p_2, \dots\}$;
- connecteurs logiques $\{\neg, \wedge, \vee, \Rightarrow\}$;
- symboles auxiliaires : parenthèses et espace.

Remarque 1.1. 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*.

Définition 1.2. L'ensemble \mathcal{F}_{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 1.3. 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 étiquetées 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 1.2.

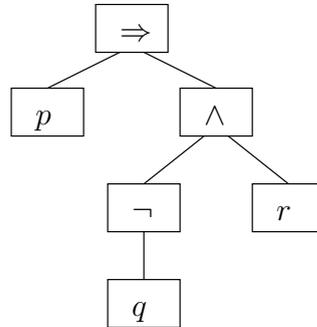


FIGURE 1.1 – Représentation arborescente de la formule $p \Rightarrow (\neg q \wedge r)$

Notation 1.4. 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 \perp 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 \perp peut être utilisé à la place de $p \wedge \neg p$ et \top à la place de $p \vee \neg p$.)

Définition 1.5 (Sous-formule). L'ensemble $SF(\varphi)$ des sous-formules d'une formule φ est défini par induction de la façon suivante.

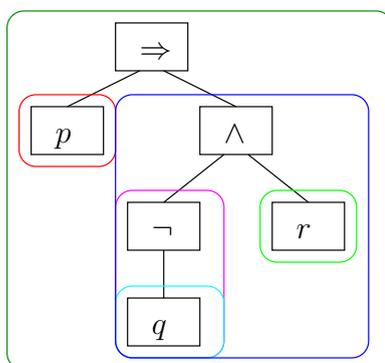
- $SF(p) = \{p\}$;
- $SF(\neg\varphi) = \{\neg\varphi\} \cup SF(\varphi)$;
- $SF(\varphi \circ \psi) = \{\varphi \circ \psi\} \cup SF(\varphi) \cup SF(\psi)$ (où \circ désigne un des symboles $\wedge, \vee, \Rightarrow$).

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 1.2).

Définition 1.6 (Sous-formule stricte). ψ est une sous-formule stricte de φ si ψ est une sous-formule de φ qui n'est pas φ .

1.3 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,

FIGURE 1.2 – Représentation arborescente des sous-formules de $p \Rightarrow (\neg q \wedge r)$

dépend donc des valeurs choisies pour les symboles propositionnels. Ces valeurs sont données par une **valuation**.

Définition 1.7 (Valuation). Une valuation est une application de PROP dans $\{0, 1\}$. La valeur 0 désigne le "faux" et la valeur 1 désigne le "vrai".

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

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 1.8 (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 circulaire : afin d'expliquer la logique, nous sommes en train de l'utiliser (*ssi, ou, et ...*). Ceci peut être contournée en utilisant la définition (équivalente à la Définition 1.8) suivante.

Définition 1.9 (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 elle-même et qu'il n'y a pas besoin de la justifier plus.

Exercice 1.10. 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 1.9 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 1.11 (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 où la prémisse est fautive correspond à 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 fautive, alors je peux démontrer des choses fautes (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 froid, 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 » ?

Terminons en mentionnant 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 1.12. Définissons l'ensemble $\text{PROP}(\varphi)$ des variables propositionnelles contenues dans $\varphi \in \mathcal{F}_{\text{cp}}$ par l'induction suivante :

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

Montrez que :

- $\text{PROP}(\varphi) = \text{PROP} \cap SF(\varphi)$, pour tout $\varphi \in \mathcal{F}_{\text{cp}}$;
- si $v(p) = v'(p)$ pour tout $p \in \text{PROP}(\varphi)$, alors $v(\varphi) = v'(\varphi)$.

Cet exercice montre que pour présenter les modèles d'une formule φ , on peut se restreindre aux valuations $v : \text{PROP}(\varphi) \rightarrow \{0, 1\}$.

1.3.1 Modèles d'une formule

Définition 1.13. L'ensemble des **valuations** d'un ensemble de variables propositionnelles PROP est noté $\text{Val}(\text{PROP})$ (ou juste Val lorsqu'il n'y a pas d'ambiguïté sur PROP). $\text{Val}(\text{PROP})$ est donc l'ensemble des fonctions de PROP dans $\{0, 1\}$.

Par exemple, si $\text{PROP} = \{p, q, r\}$, alors Val est représenté par la Table 1.1, dans laquelle chaque ligne est une valuation de 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 1.1 – Val pour PROP = $\{p, q, r\}$

Définition 1.14 (Modèle d'une formule). Un **modèle** de φ est une valuation v telle que $v(\varphi) = 1$. On note $\text{mod}(\varphi)$ l'ensemble des modèles de φ .

Exemple 1.15. Si PROP = $\{p, q, r\}$ et $\varphi = (p \vee q) \wedge (p \vee \neg r)$ alors l'ensemble des modèles de φ est

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

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

Définition 1.16 (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 $\text{mod}(\varphi) \neq \emptyset$).

Définition 1.17 (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 $\text{mod}(\varphi) = \emptyset$).

Définition 1.18 (Tautologie). Une formule φ est une **tautologie** (ou **valide**) si $v(\varphi) = 1$ pour toute valuation v (*i.e.*, si $\text{mod}(\varphi) = \text{Val}$). 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 1.19. 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 1.20 (Equivalence). On dit que φ est **équivalente à** ψ si les deux formules ont les mêmes modèles (*i.e.* si $\text{mod}(\varphi) = \text{mod}(\psi)$). On note alors $\varphi \equiv \psi$.

Exemple 1.21. 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.*, $\text{mod}(\varphi) = \text{mod}(\varphi[\psi \leftarrow \psi'])$.

Exercice 1.22. Prouvez les équivalences suivantes :

$\varphi \vee \perp \equiv \varphi$	$\varphi \wedge \perp \equiv \perp$	$(\varphi \wedge \psi) \wedge \theta \equiv \varphi \wedge (\psi \wedge \theta)$
$\varphi \vee \top \equiv \top$	$\varphi \wedge \top \equiv \varphi$	$(\varphi \wedge \psi) \vee \theta \equiv (\varphi \vee \theta) \wedge (\psi \vee \theta)$
$\varphi \vee \psi \equiv \psi \vee \varphi$	$\varphi \wedge \psi \equiv \psi \wedge \varphi$	$(\varphi \vee \psi) \wedge \theta \equiv (\varphi \wedge \theta) \vee (\psi \wedge \theta)$
$\varphi \vee \varphi \equiv \varphi$	$\neg(\varphi \vee \psi) \equiv (\neg\varphi) \wedge (\neg\psi)$	$(\varphi \vee \psi) \vee \theta \equiv \varphi \vee (\psi \vee \theta)$
$\neg\neg\varphi \equiv \varphi$	$\varphi \wedge \varphi \equiv \varphi$	$\neg(\varphi \wedge \psi) \equiv (\neg\varphi) \vee (\neg\psi)$

La proposition suivante explicite les liens forts existants entre logique et algèbre ensembliste. En fait, par la sémantique, nous sommes en train d'étudier et comprendre la logique par cette algèbre (dont nous supposons disposer d'une compréhension intuitive et exacte).

Proposition 1.23. Soient φ et ψ deux formules, on a :

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

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

$$\begin{aligned} v \in \text{mod}(\neg\varphi) \text{ ssi } v(\neg\varphi) = 1 & \qquad \text{par la définition de modèle} \\ \text{ssi } v(\varphi) = 0 & \qquad \text{par la Définition 1.8} \\ \text{ssi } v \notin \text{mod}(\varphi) & \qquad \text{encore, par la définition de modèle} \\ \text{ssi } v \in \text{Val} - \text{mod}(\varphi) & \end{aligned}$$

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

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

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

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

4.

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

Définition 1.24 (Conséquence logique). Une formule φ est **conséquence logique** d'une formule

ψ si tout modèle de ψ est un modèle de φ (c'est-à-dire, si $\text{mod}(\psi) \subseteq \text{mod}(\varphi)$). On note alors $\psi \models \varphi$.

Remarque 1.25. Attention à la confusion dans les notations !

- $\psi \models \varphi$ où ψ est une formule : φ est conséquence logique de ψ ;
- $\models \varphi$ (rien à gauche du symbole \models) : φ est une tautologie ;
- $v \models \varphi$ où v est une valuation, i.e., l'assignation d'une valeur aux propositions atomiques de la formule ; c'est un raccourcis assez fréquent dans la littérature (noté quelques-fois aussi $v \Vdash \varphi$, exactement pour ne pas utiliser \models) pour $v(\varphi) = 1$. Dans ces notes, nous essayerons d'éviter cette notation, si possible.

Attention : la notation $\psi \models \varphi$ signifie que φ est conséquence logique de ψ (donc $\text{mod}(\psi) \subseteq \text{mod}(\varphi)$) et non pas que ψ est conséquence logique de φ , ce que serait plutôt $\text{mod}(\varphi) \subseteq \text{mod}(\psi)$.

Proposition 1.26. 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 1.23
- 2.

$$\begin{aligned} \varphi \equiv \psi \text{ ssi } \text{mod}(\varphi) &= \text{mod}(\psi) \\ &\text{ssi } \text{mod}(\varphi) \subseteq \text{mod}(\psi) \text{ et } \text{mod}(\psi) \subseteq \text{mod}(\varphi) \\ &\text{ssi } \models \varphi \Rightarrow \psi \text{ et } \models \psi \Rightarrow \varphi \\ &\text{ssi } \models \varphi \Rightarrow \psi \wedge \psi \Rightarrow \varphi. \end{aligned}$$

□

1.3.2 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 1.27 (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 $\text{mod}(\Gamma)$ l'ensemble des modèles de Γ .

Cet ensemble de modèles est donc l'ensemble des valuations qui respectent toutes les contraintes de Γ .

Définition 1.28 (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 $\text{mod}(\Gamma) \neq \emptyset$);
- **insatisfaisable** (ou **contradictoire**, ou **inconsistant**, ou encore **incohérent**) s'il n'admet aucun modèle (*i.e.*, si $\text{mod}(\Gamma) = \emptyset$), on note alors $\Gamma \models \perp$.

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

Définition 1.29 (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 $\text{mod}(\Gamma) \subseteq \text{mod}(\varphi)$), on note alors $\Gamma \models \varphi$. On note $\text{cons}(\Gamma)$ l'ensemble des conséquences logiques de Γ .

Remarque 1.30. Attention aux deux notations :

- $\psi \models \varphi$ où ψ est une formule ;
- $\Gamma \models \varphi$ où Γ est un ensemble de formule.

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

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

Proposition 1.31. $\Gamma \models \varphi$ ssi $\Gamma \cup \{\neg\varphi\}$ est contradictoire.

Démonstration. On peut calculer comme suit :

- $\Gamma \models \varphi$ 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 1.32. Pour tous ensembles de formules Σ, Γ ,

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

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

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 1.32. Pour toute valuation v :

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

La preuve de la Proposition suivante est laissée en exercice.

Proposition 1.33. Si $\Gamma' \subseteq \Gamma$ et $\Gamma' \models \varphi$, alors $\Gamma \models \varphi$.

Proposition 1.34. Soit $\Gamma = \{\varphi_1, \dots, \varphi_n\}$ un ensemble fini de formules. Nous avons alors

$$\text{mod}(\Gamma) = \text{mod}(\varphi_1 \wedge \dots \wedge \varphi_n),$$

de façon que $\{\varphi_1, \dots, \varphi_n\} \models \psi$ ssi $\varphi_1 \wedge \dots \wedge \varphi_n \models \psi$, ssi $(\varphi_1 \wedge \dots \wedge \varphi_n) \Rightarrow \psi$ 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 peut pas se généraliser (au moins, de façon immédiate) au cas des ensembles de formules de taille infinie. Dans le cas où Γ est infini, il faudra utiliser le théorème de compacité (Théorème 1.40) 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

$$\begin{aligned} \text{mod}(\{\varphi_1, \dots, \varphi_n\}) &= \text{mod}(\{\varphi_1\} \cup \dots \cup \{\varphi_n\}) \\ &= \text{mod}(\{\varphi_1\}) \cap \dots \cap \text{mod}(\{\varphi_n\}) && \text{(par la Proposition 1.32)} \\ &= \text{mod}(\varphi_1) \cap \dots \cap \text{mod}(\varphi_n) && \text{(par la Remarque 1.30)} \\ &= \text{mod}(\varphi_1 \wedge \dots \wedge \varphi_n). && \text{(par la Proposition 1.23)} \end{aligned}$$

Donc, on a que $\text{mod}(\{\varphi_1, \dots, \varphi_n\}) \subseteq \text{mod}(\psi)$ si et seulement si $\text{mod}(\varphi_1 \wedge \dots \wedge \varphi_n) \subseteq \text{mod}(\psi)$ et, par la Proposition 1.23, la dernière relation est vraie ssi $(\varphi_1 \wedge \dots \wedge \varphi_n) \Rightarrow \psi$ est une tautologie. \square

Proposition 1.35. $\Gamma \models \varphi$ si, et seulement si, $\text{mod}(\Gamma) = \text{mod}(\Gamma \cup \{\varphi\})$.

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 1.35. La proposition découle du fait que $\text{mod}(\Gamma \cup \{\varphi\}) = \text{mod}(\Gamma) \cap \text{mod}(\{\varphi\})$, et que la relation $\text{mod}(\Gamma) \subseteq \text{mod}(\varphi)$ est équivalente à $\text{mod}(\Gamma) \cap \text{mod}(\{\varphi\}) = \text{mod}(\Gamma)$. \square

Exemple 1.36. 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 1.35 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, $\text{mod}(\Gamma) = \text{mod}(\Gamma - \{\varphi\})$. L'exemple suivant éclaircit cette méthode.

Exemple 1.37. 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, q, r, s , qui obéissent aux contraintes suivantes :

$$\begin{aligned}\varphi_1 &: q \wedge \neg r \\ \varphi_2 &: p \Rightarrow (r \vee s) \\ \varphi_3 &: \neg r \wedge (q \vee p)\end{aligned}$$

Soit $\Gamma_1 = \{\varphi_1, \varphi_2, \varphi_3\}$, cet ensemble forme l'ensemble des contraintes sur 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 $v(r) = 0$ et $v(q) = 1$,
- v satisfait φ_3 ssi $v(r) = 0$ et ($v(p) = 1$ ou $v(q) = 1$).

D'où, $\text{mod}(\varphi_1) \subseteq \text{mod}(\varphi_3)$.

(b) Soit $\Gamma_2 = \{\varphi_1, \varphi_2\}$, on a $\text{mod}(\Gamma_2) = \text{mod}(\Gamma_1)$. En effet, par définition,

$$\text{mod}(\Gamma_1) = \text{mod}(\varphi_1) \cap \text{mod}(\varphi_2) \cap \text{mod}(\varphi_3) = \text{mod}(\varphi_1) \cap \text{mod}(\varphi_2) = \text{mod}(\Gamma_2).$$

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	q	r	s
0	1	0	0
0	1	0	1
1	1	0	0
1	1	0	1

Modèles de φ_2 :

p	q	r	s
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'intersection des modèles de φ_1 et φ_2 :

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

(Évidemment, nous aurions pu éviter de calculer tous les modèles de φ_2 !!!)

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 :

$$\neg r, q, q \wedge \neg r \in \text{cons}(\Gamma)$$

5. Ajoutons maintenant une nouvelle contrainte : $\Gamma_3 = \{\neg q \wedge \neg s\} \cup \Gamma_2$. On a $\text{mod}(\Gamma_3) = \text{mod}(\Gamma_2) \cap \text{mod}(\neg q \wedge \neg s) = \emptyset$. Donc $\text{mod}(\Gamma_3) = \emptyset$ et Γ_3 est contradictoire.

1.3.2.1 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 crucial plus tard, dans le cadre de la preuve de complétude pour le calcul de la résolution (Théorème ??).

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

Lemme 1.38 (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 $e_0, e_1, \dots, e_n, \dots$ dans T par induction sur les entiers. La propriété suivante :

$$P(n) ::= \text{le sous-arbre issu de } e_n \text{ est infini}$$

sera 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 avoir construit e_0, \dots, e_n et que $P(n)$ est vraie, c'est-à-dire que le sous arbre issu de e_n est infini. Considérons les successeurs immédiats de e_n , appelons-les e_{n_1}, \dots, e_{n_k} ; si tous étaient racines de sous-arbres finis, de cardinalité p_1, \dots, p_k , alors il en serait de même de e_n (le sous-arbre issu de e_n aurait alors cardinalité $1 + p_1 + \dots + p_k$), contradiction. Donc l'un d'entre eux est racine d'un sous-arbre infini, ce noeud étant le e_{n+1} cherché. \square

Nous aurons besoin du Lemme dans la forme suivante :

Lemme 1.39. *Tout arbre a branchement fini et dont toutes les branches sont finies, est fini.*

Théorème 1.40 (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 1.41 (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 (du Théorème 1.41). On fait la preuve dans le cas où $\text{PROP} = \{p_0, p_1, p_2, \dots, p_n, \dots\}$ 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 un symbole propositionnel. La racine (le niveau 0) est associée 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 π partant de la racine correspond à une valuation v_π de l'ensemble des propositions ;
2. chaque noeud e à profondeur n correspond à une valuation v_e des variables $\{p_0, \dots, p_{n-1}\}$.

Nous appelons un noeud e de l'arbre *noeud d'échec* (par rapport à Γ) s'il existe une formule $\varphi_e \in \Gamma$ telle que $\text{PROP}(\varphi_e) \subseteq \{p_0, \dots, p_{n-1}\}$ 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 inconsistent.

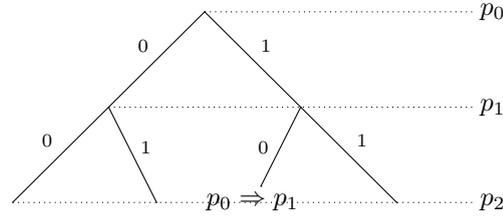


FIGURE 1.3 – Début de l’arbre sémantique avec le noeud d’échec 10 étiqueté

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’échec. (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 $v \in \text{Val}$, alors $v = v_\pi$ pour une branche π de l’arbre, et donc $v(\varphi_{e(\pi)}) = v_\pi(\varphi_{e(\pi)}) = v_{e(\pi)}(\varphi_{e(\pi)}) = 0$.) L’ensemble $\Gamma_0 \subseteq \Gamma$ est donc fini et inconsistant. \square

Exemple 1.42. Supposons que Γ soit de la forme $\{ p_0 \wedge \neg p_1, p_0 \Rightarrow p_1, \dots \}$. 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 début de) l’arbre sémantique, avec le noeud 10 étiqueté par la formule témoignant son échec, est représenté en Figure 1.3.

Remarque 1.43. 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 1.44. Une formule φ est conséquence d’un ensemble de formules Γ si et seulement s’il existe un sous-ensemble fini $\Gamma_{f\text{ini}}$ de Γ tel que $\Gamma_{f\text{ini}} \models \varphi$.

Démonstration.

$\Gamma \models \varphi$ ssi $\Gamma \cup \{ \neg \varphi \}$ est contradictoire par la Proposition 1.31
 ssi il existe $\Gamma_f \subseteq \Gamma \cup \{ \neg \varphi \}$ fini et contradictoire par le Théorème de compacité
 ssi il existe $\Gamma_f \subseteq \Gamma$ fini tel que $\Gamma_f \cup \{ \neg \varphi \}$ est contradictoire
 ssi il existe un sous-ensemble fini $\Gamma_f \subseteq \Gamma$ tel que $\Gamma_f \models \varphi$. \square

1.3.3 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 1.45. 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 le cardinal de $\text{PROP}(\varphi)$ (nombre de symboles propositionnels dans φ). \square

Nous verrons par la suite qu'il y a de meilleurs algorithmes, mais qu'ils 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.

1.4 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.

1.4.1 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 les formules $\varphi = (\neg p \vee q) \wedge (\neg p \vee \neg r)$, $\psi = \neg p$ 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 1.46 (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 1.47. 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.

1.4.2 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.*, $\text{mod}(\varphi) = \text{mod}(\varphi_{[\psi \leftarrow \psi']})$.

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)
$\top \wedge \varphi \equiv \varphi \wedge \top \equiv \varphi$	$\perp \vee \varphi \equiv \varphi \vee \perp \equiv \varphi$	(Éléments 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)
$\varphi \wedge \perp \equiv \perp \wedge \varphi \equiv \perp$	$\varphi \vee \top \equiv \top \vee \varphi \equiv \top$	(Élément 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)
$\varphi \wedge \neg \varphi \equiv \neg \varphi \wedge \varphi \equiv \perp$	$\varphi \vee \neg \varphi \equiv \neg \varphi \vee \varphi \equiv \top$	(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 :

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

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

Observez que, dans les exemples d'équivalences ci-dessus, si $\varphi \equiv \psi$ et φ, ψ sont des formules sans implication, alors on a aussi l'équivalence $\varphi' \equiv \psi'$ où φ' et ψ' sont les formules obtenues de φ et ψ en échangeant la conjonction avec la disjonction et vice-versa (donc, en échangeant aussi le vrai par le faux). C'est un principe tout à fait général que l'on peut énoncer ici :

Proposition 1.48 (Principe de dualité). *Si $\varphi \equiv \psi$ où φ et ψ sont des formules sans implication, alors $\varphi' \equiv \psi'$ où φ' et ψ' sont les formules obtenues de φ et ψ en échangeant la conjonction avec la disjonction et vice-versa.*

Exemple 1.49. Nous pouvons argumenter que $\varphi \wedge (\psi \vee \chi) \equiv \varphi \wedge (\chi \vee \psi)$ de façon précise de cette façon :

$$\begin{aligned} \varphi \wedge (\psi \vee \chi) &\equiv (\varphi \wedge p)[p \leftarrow \psi \vee \chi] \\ &\equiv (\varphi \wedge p)[p \leftarrow \chi \vee \psi] \quad (\text{par commutativité, et en utilisant la proposition 1.47}) \\ &\equiv \varphi \wedge (\chi \vee \psi). \end{aligned}$$

1.4.3 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 formules possibles construites à partir des formules $\varphi_1, \dots, \varphi_n$ 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. On peut faire la remarque analogue pour le connecteur logique \vee .

Ainsi, si $\varphi_1, \dots, \varphi_n$ sont des formules, nous pouvons utiliser les notations :

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

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

Nous pouvons même étendre ces considérations plus loin : si ψ possède plus que deux occurrences dans la liste $\varphi_1, \dots, \varphi_n$, 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 équivalentes.

La conclusion de l'ensemble de ces remarques est la suivante : puisque ni l'ordre, ni le parenthésage, ni la multiplicité n'influent sur la sémantique de la formule, 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 répétitions) ; la longueur n de la liste pourra aussi prendre les valeurs 0 et 1, en posant :

$$\begin{aligned} \bigwedge_{i=1}^0 \varphi_i &:= \top, & \bigvee_{i=1}^0 \varphi_i &:= \perp, & \text{pour } n = 0, \\ \bigwedge_{i=1}^1 \varphi_i &:= \varphi_1, & \bigvee_{i=1}^1 \varphi_i &:= \varphi_1, & \text{pour } n = 1. \end{aligned}$$

Définition 1.50 (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 1.51 (Clause). Une **clause disjunctive** 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.

Remarque 1.52. Comme d'usage, quand le mot « clause » est utilisé sans spécifications ultérieures, l'adjectif « disjunctive » est sous-entendu. Ainsi, *clause* tout-court est synonyme de *clause disjunctive*.

Définition 1.53 (Forme normale conjonctive). Une **formule conjonctive** (ou formule sous forme normale conjonctive (FNC), ou sous forme clausale) est une conjonction de clauses disjunctives : $\bigwedge_{j=1}^m C_j$ où les C_j sont des clauses disjunctives, ou encore $\bigwedge_{j=1}^m \bigvee_{i=1}^{n_j} \ell_i^j$ où les ℓ_i^j sont des littéraux.

Exemple 1.54. La formule suivante est sous forme clausale :

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

Définition 1.55 (Forme normale disjunctive). Une **formule disjunctive** (ou formule sous forme normale disjunctive (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 1.56. La formule suivante est sous forme normale disjunctive :

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

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 (le Lemme 1.57 ci-dessous précise ce point). Dans l'exemple 1.54, 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.

Lemme 1.57. Soit φ une formule en FNC : $\varphi = \bigwedge_{i=1, \dots, n} C_i$ où C_i sont des clauses. Pour tout $v \in \text{Val}$, $v \in \text{mod}(\varphi)$ ssi, pour tout $i = 1, \dots, n$, il existe un littéral l in C_i tel que $v(l) = 1$.

La preuve de ce lemme étant immédiate, elle est laissée au lecteur.

1.4.3.1 Mise sous forme clausale, en préservant l'équivalence

L'algorithme consiste en l'application des 3 étapes suivantes :

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) :

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

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 1.58. Considérons $\varphi = (\neg(p \wedge (q \Rightarrow (r \vee s)))) \wedge (p \vee q)$.

- Etape 1 :
Remplacer $q \Rightarrow (r \vee s)$ par $\neg q \vee (r \vee s)$ dans φ :
 $\varphi \equiv (\neg(p \wedge (\neg q \vee (r \vee s)))) \wedge (p \vee q)$
- Etape 2 :
 $\varphi \equiv (\neg p \vee \neg(\neg q \vee (r \vee s))) \wedge (p \vee q)$
 $\varphi \equiv (\neg p \vee (\neg\neg q \wedge \neg(r \vee s))) \wedge (p \vee q)$
 $\varphi \equiv (\neg p \vee (q \wedge \neg(r \vee s))) \wedge (p \vee q)$
 $\varphi \equiv (\neg p \vee (q \wedge \neg r \wedge \neg s)) \wedge (p \vee q)$
- Etape 3 :
 $\varphi \equiv (\neg p \vee q) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee \neg s) \wedge (p \vee q)$

Proposition 1.59. *Le calcul précédent termine et donne une formule en forme clausale équivalente à la formule initiale.*

Remarque 1.60. Il n'y a pas unicité de la forme clausale.

1.4.3.2 Mise sous forme clausale, en préservant la satisfaisabilité

Considérons une formule φ du calcul propositionnel ayant ψ comme sous-formule : on peut toujours considérer que φ de la forme $\varphi'[p \leftarrow \psi]$, pour un p choisi tel que $p \notin \text{PROP}(\varphi)$.

Proposition 1.61. *Soit $\varphi, \psi \in \mathcal{F}_{cp}$ and soit $p \notin \text{PROP}(\psi)$; supposons que φ ne contient pas de négations. On a alors*

$$\text{mod}(\varphi[p \leftarrow \psi]) \neq \emptyset \text{ ssi } \text{mod}(\varphi \wedge (p \Rightarrow \psi)) \neq \emptyset.$$

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

Démonstration. Soit v une valuation telle que $v(\varphi[p \leftarrow \psi]) = 1$, et v' la valuation satisfaisant, $v'(q) = v(p)$ pour tout $q \neq p$ et $v'(p) = v(\psi)$. On a alors $v'(\varphi \wedge (p \Rightarrow \psi)) = 1$. On a donc $\text{mod}(\varphi[p \leftarrow \psi]) \neq \emptyset$ implique $\text{mod}(\varphi \wedge (p \Rightarrow \psi)) \neq \emptyset$.

Nous montrons l'implication inverse par induction sur la structure de φ : on prouve que si v est un modèle de $\varphi \wedge (p \Rightarrow \psi)$ alors v est un modèle de $\varphi[p \leftarrow \psi]$.

Avant de débiter l'induction, remarquons que si $v(\varphi \wedge p \Rightarrow \psi) = 1$, alors $v(\varphi) = 1$ et $v(p) \leq v(\psi)$.

Induction :

- cas de base :
 - si $\varphi = q$ où $q \neq p$ est un symbole propositionnel, alors $v(\varphi[p \leftarrow \psi]) = q = 1$;
 - si $\varphi = p$, alors $v(p) = 1$ (car $v(\varphi) = 1$) et donc $v(\psi) = 1$ (car par hypothèse $v(p) \leq v(\psi)$). On obtient donc $v(\varphi[p \leftarrow \psi]) = v(\psi) = 1$;
- pas d'induction : supposons que la propriété est vraie pour deux formules φ_1 et φ_2 .
 - si $\varphi = \varphi_1 \wedge \varphi_2$, alors puisque $v(\varphi) = 1$, on a forcément $v(\varphi_1) = 1$ et $v(\varphi_2) = 1$. Puisqu'on a également supposé que $v(p) \leq v(\psi)$, $v(\varphi_1 \wedge p \Rightarrow \psi) = 1$ et $v(\varphi_2 \wedge p \Rightarrow \psi) = 1$. Appliquons l'hypothèse d'induction à φ_1 et φ_2 , nous obtenons $v(\varphi_1[p \leftarrow \psi]) = 1$ et $v(\varphi_2[p \leftarrow \psi]) = 1$. En conclusion, $v(\varphi[p \leftarrow \psi]) = \min\{v(\varphi_1[p \leftarrow \psi]), v(\varphi_2[p \leftarrow \psi])\} = 1$
 - le cas $\varphi = \varphi_1 \vee \varphi_2$ est similaire au précédent. Les cas $\varphi = \neg\varphi_1$ et $\varphi = \varphi_1 \Rightarrow \varphi_2$ sont interdits par l'hypothèse que φ ne contient pas la négation (la négation apparaît implicitement dans l'implication).

□

Exercice 1.62. Montrez que l'hypothèse que la négation n'apparaît pas dans φ est nécessaire.

Exercice 1.63. Montrez que les deux formules $\varphi[p \leftarrow \psi]$ et $\varphi \wedge (p \Rightarrow \psi)$ ne sont pas équivalentes.

Exemple 1.64. Considérez la formule

$$\bigvee_{i=1}^n p_{i,0} \wedge p_{i,1}. \quad (1.1)$$

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

$$\bigwedge_{f:[1,n] \rightarrow \{0,1\}} p_{1,f(1)} \vee p_{2,f(2)} \vee \dots \vee p_{n,f(n)}.$$

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

La Proposition 1.61 montre que la formule (1.1) est équisatisfaisable avec la formule

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

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

La Proposition 1.61 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 modèle, et non pas à énumérer tous les modèles. Parmi les défauts de la méthode, on peut noter que le fait introduire des nouveaux symboles propositionnels alourdi évidemment les calculs de recherche d'un modèle.

1.5 Le problème SAT

1.5.1 Définition du problème

Définition 1.65 (Problème SAT). Le problème SAT est le problème de décision qui consiste à déterminer si $\varphi \in \mathcal{F}_{cp}$ 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, ils peuvent fournir aussi un modèle, ou même l'ensemble des modèles.

Exemple 1.66. 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 $v \in \text{mod}(\varphi)$.

1.5.2 Un problème NP-complet

Théorème 1.67. *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. \square

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 satisferaient 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 ??).

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.

1.5.3 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).

1.5.3.1 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 faus.

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 , p seulement si q	$p \Rightarrow q$
p si et seulement si q	$p \Leftrightarrow q$

Exemple 1.68 (suite).

Si et seulement si : Considérons l'énoncé

« J'irai à la plage si, et seulement si, il y a du soleil. »

et posons :

- *plage* : j'irai à la plage
- *soleil* : il y a du soleil

L'énoncé se traduit par $plage \Leftrightarrow soleil$. En effet, décomposons l'énoncé :

- « J'irai à la plage si il y a du soleil » se traduit par $soleil \Rightarrow plage$.
- « J'irai à la plage seulement si il y a du soleil » signifie que si je vais à la plage, alors il a forcément du soleil et se traduit donc par $plage \Rightarrow soleil$.

On a donc $soleil \Rightarrow plage \wedge plage \Rightarrow soleil$, ce qui est équivalent à $soleil \Leftrightarrow plage$.

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 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 :

<i>note</i>	<i>travail</i>	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 :

<i>note</i>	<i>travail</i>	P_2
1	1	1
1	0	1
0	0	1
0	1	0

Donc $P_2 \equiv travail \Rightarrow note$.

1.5.3.2 Formalisation du problème du Sudoku pour la réduction à SAT

Soit ℓ un entier, et $n\ell^2$. Une grille de Sudoku de taille $n \times n$, divisée en une grille $\ell \times \ell$ de façon à former n blocs. Généralement, la grille a les dimensions $n = 9$ et $\ell = 3$. Au départ certaines cases sont remplies par des nombres compris entre 1 et n , et d'autres sont vides. Le but du jeu est de remplir les cases vides avec des nombres dans $[1, n]$ en respectant les règles suivantes :

- On ne doit pas avoir deux nombres identiques sur une même ligne ;
- On ne doit pas avoir deux nombres identiques sur une même colonne ;
- Il ne doit pas y avoir deux nombres identiques dans l'un des blocs.

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 $x, y, z \in [1, n]$. 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 nombre » : 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 $[1, n]$: la case (i, j) ne contient pas k' .

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

« On ne doit pas avoir deux nombres identiques sur une même colonne » : pour toute colonne $i \in [1, n]$, pour toute ligne $j \neq j'$ 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 nombres identiques sur une même ligne » : pour toute ligne $j \in [1, n]$, pour toute colonne $i \neq i'$ 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 nombres identiques dans le même bloc » : 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.

1.5.4 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 1.69. Nous pouvons identifier une formule en forme normale conjonctive φ avec l'ensemble \mathcal{C}_φ des ses clauses, car évidemment nous avons $\text{mod}(\varphi) = \text{mod}(\mathcal{C}_\varphi)$ (voir la Proposition ??). 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, étant donné un ensemble de clauses \mathcal{C} , nous pouvons poser

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

de sorte que $\mathcal{C} = \mathcal{C}_\varphi$. Les algorithmes que nous allons considérer manipulent plutôt des ensembles des clauses, que des formules en FNC.

Remarque 1.70 (Simplifications). Avant chercher de 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 : Etant données deux clauses C et C' , on dit que C *subsume* C' si C est incluse dans C' , c'est-à-dire, si tout littéral apparaissant dans C apparaît aussi dans C' .

Si un ensemble de clauses \mathcal{C} contient deux clauses C et C' telles que C subsume C' , alors la clause C' peut être supprimée de \mathcal{C} . En effet, on a dans ce cas $\mathcal{C} \models C'$ et donc $\text{mod}(\mathcal{C}) = \text{mod}(\mathcal{C} \setminus \{C_j\})$, par la Proposition ??.

Par exemple $C_i = p \vee q \vee r$ est incluse dans $C_j = p \vee \neg s \vee t \vee q \vee r$, donc on peut supprimer C_i d'une formule clausale de la forme $C_1 \wedge \dots \wedge C_i \wedge \dots \wedge C_j \wedge \dots \wedge C_n$.

Remarque 1.71 (Substitutions simples). Soit ψ une formule sous forme clausale, et \mathcal{C} l'ensemble de ses clauses. Le calcul d'une formule équivalente à $\psi[p \leftarrow \varphi]$, où $\varphi \in \{\perp, \top\}$, peut se faire aisément via une procédure purement syntaxique sur l'ensemble \mathcal{C} . Définissons cette procédure :

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

$\mathcal{C}[p \leftarrow \perp]$: 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

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

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 $\mathcal{C}[p \leftarrow \perp]$. C'est donc la substitution qui force ce littéral à être vrai. Nous utiliserons la notation $\mathcal{C}[\ell \leftarrow \perp]$ pour $\mathcal{C}[\bar{\ell} \leftarrow \perp]$, $\bar{\ell}$ est le littéral opposé de ℓ : $\bar{\ell} = \neg p$ si $\ell = p$, et $\bar{\ell} = p$ si $\ell = \neg p$.

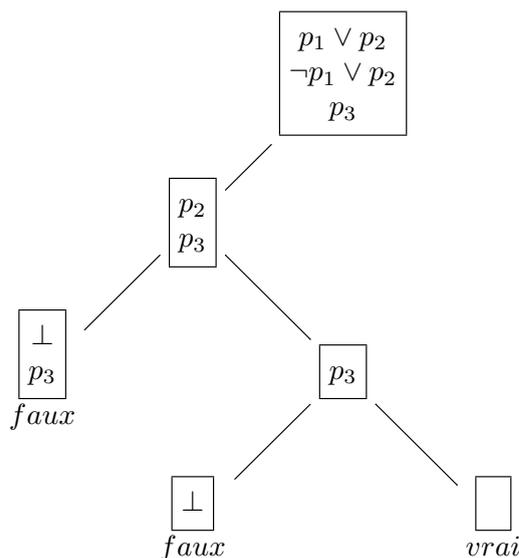
1.5.4.1 Algorithme de Quine

Nous présentons d'abord la méthode de Quine dans le cas restreint de formules mises au préalable sous forme normale conjonctive, assimilées donc à des ensembles 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 ??). Chaque fois qu'une variable est affectée à la valeur $x \in \{0, 1\}$, le calcul se fait récursivement avec $\mathcal{C}[p \leftarrow \perp]$, si $x = 0$, et avec $\mathcal{C}[p \leftarrow \top]$, sinon.

Algorithme de Quine
entrée : un ensemble de clauses \mathcal{C}
sortie : <i>vrai</i> si \mathcal{C} est satisfaisable ou <i>faux</i> sinon
simplifier l'ensemble de clauses (cf. Remarque 1.70) ;
si $\mathcal{C} = \emptyset$ retourner <i>vrai</i>
si \mathcal{C} contient la clause \perp retourner <i>faux</i>
choisir le prochain $p \in \text{PROP}$ apparaissant dans une clause
si $\text{Quine}(\mathcal{C}[p \leftarrow \perp]) = \text{vrai}$ alors retourner vrai
sinon retourner $\text{Quine}(\mathcal{C}[p \leftarrow \top])$

Exemple 1.72. 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 amèneront 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 de 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.

1.5.4.2 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 plupart des outils de satisfaction de contraintes

(« SAT solvers », en anglais) implémentent une variante de cet algorithme. Il peut être vu comme un raffinement de la méthode de Quine ; l'amélioration principale qu'il apporte est

- la réduction du branchement via la propagation des clauses unitaires ;
- l'utilisation d'heuristiques pour accélérer le parcours des solutions.

Algorithme <i>DPLL</i>	
entrée : un ensemble de clauses \mathcal{C}	
sortie : <i>vrai</i> si \mathcal{C} est satisfaisable ou <i>faux</i> sinon	
simplifier l'ensemble de clauses (cf. Remarque 1.70) ;	
si $\mathcal{C} = \emptyset$ retourner <i>vrai</i>	
si \mathcal{C} contient la clause \perp retourner <i>faux</i>	
si \mathcal{C} contient la clause unitaire ℓ retourner $DPLL(\mathcal{C}[\ell \leftarrow \top])$	(propagation des contraintes boo- léennes)
choisir un littéral ℓ depuis une clause et $x, y \in \{\top, \perp\}$, $x \neq y$ avec la bonne heuristique !	(décision des lit- téraux)
si $DPLL(\mathcal{C}[\ell \leftarrow x]) = \textit{vrai}$ alors retourner vrai	
sinon retourner $DPLL(\mathcal{C}[\ell \leftarrow y])$	(backtracking)

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. Si une variable propositionnelle apparaît seulement sous forme positive ou seulement sous forme négative alors ses littéraux sont dits purs. On choisit un littéral pur ℓ parmi ceux qui apparaissent dans le plus de clauses, et on choisit x tel que $\ell[\ell \leftarrow x] = \top$.

Littéraux fréquents. On appelle cette heuristique DLIS, acronyme pour l'anglais « dynamic largest individual sum of literals ». Elle consiste à choisir un littéral parmi ceux apparaissant les plus ; choisir $x = \top$.

Littéraux "courts". On appelle cette heuristique MOM's, acronyme pour l'anglais « maximum occurrence in clauses of minimum size ». Elle consiste à choisir un littéral parmi ceux apparaissant les plus, dans les clauses les plus courtes ; choisir $x = \perp$.

Fréquence des variables. Choisir un symboles propositionnels parmi ceux qui apparaissent le plus, dans les clauses les plus courtes ; choisir $x = \perp$.

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 ,

$$\text{rank}_{\text{lit}}(\ell, \mathcal{C}) = \sum_{\ell \in \mathcal{C}, \mathcal{C} \in \mathcal{C}} \frac{1}{2^{\text{card} \mathcal{C}}},$$

et, pour un symbole propositionnel p ,

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

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

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

pu tout simplement définir

$$\begin{aligned} \text{rank}_{\text{lit}}(\ell, \mathcal{C}) &= \text{card}\{C \in \mathcal{C} \mid \text{card}C \text{ est minimum dans l'ensemble } \{\text{card}C \mid C \in \mathcal{C}\}\} \\ \text{rank}_{\text{lit}}(\ell, \mathcal{C}) &= \alpha \cdot \text{card}\{C \mid \ell \in C\} + \beta \cdot \frac{\text{card}\{C \mid \ell \in C\}}{\sum_{\ell \in C} \text{card}C}, \end{aligned}$$

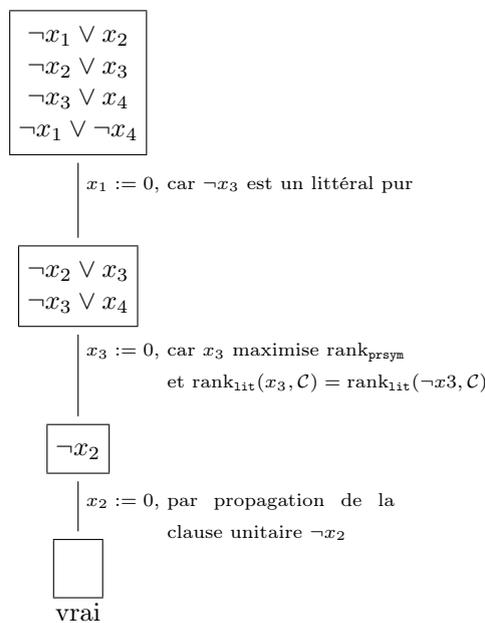
ou, si l'on souhaite aussi prendre en compte la fréquence du littéral opposé $\neg\ell$,

$$\text{rank}_{\text{lit}}(\ell, \mathcal{C}) = \alpha \cdot \text{card}\{C \mid \neg\ell \in C\} + \beta \cdot \sum_{\ell \in C, C \in \mathcal{C}} \frac{1}{2^{\text{card}C}},$$

pour des pondérations α et β bien choisies.

Remarque 1.73. Pour qu'une heuristique soit effectivement implantée dans un SAT-solveur, il faut disposer de structures de données permettant de calculer le littéral choisi de façon efficace. Ainsi, certaines heuristiques décrites ci-dessus (en particulier, celle du littéral pur) ne sont pas implémentées dans les solveurs existants les plus avancés (et performants). Voir [Sil99] pour une comparaison des heuristiques dans les SAT-solveurs.

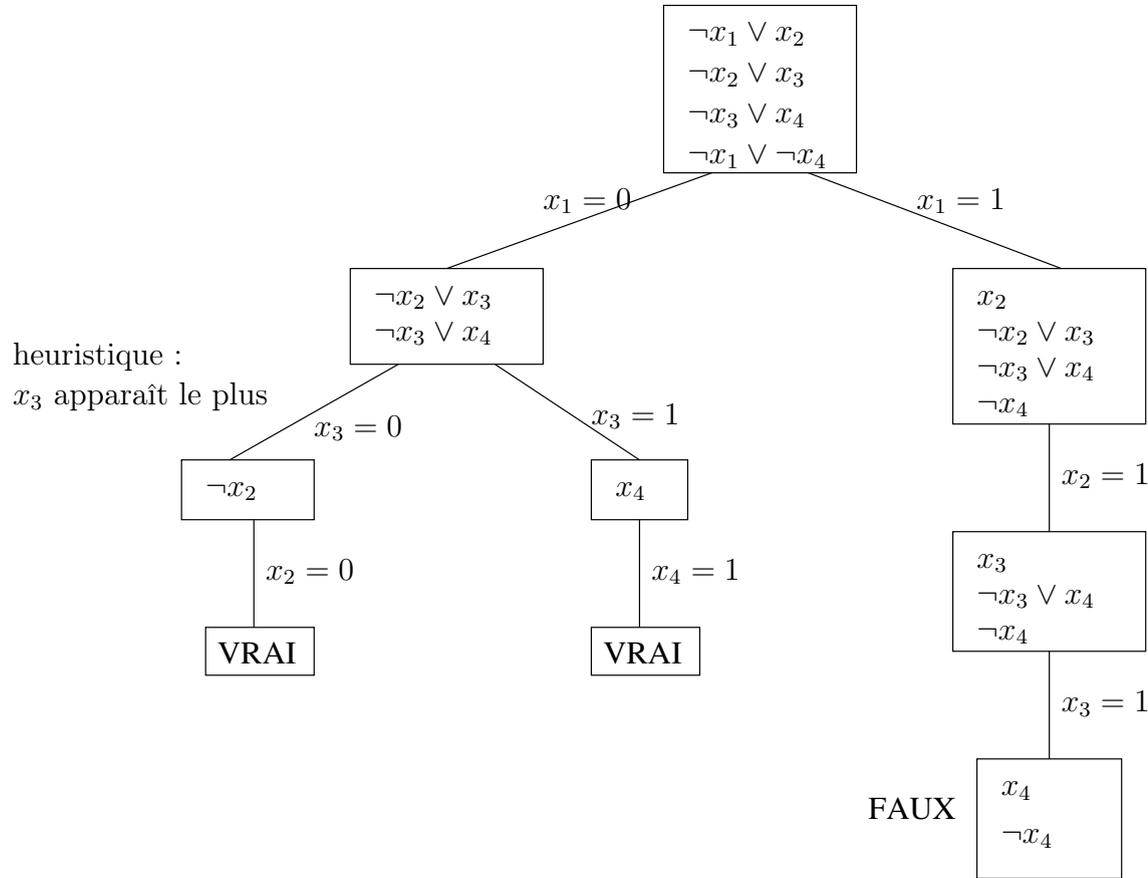
Exemple 1.74. Considérons l'ensemble de clauses $\mathcal{C} = \{\neg x_1 \vee x_2, \neg x_2 \vee x_3, \neg x_3 \vee x_4, \neg x_1 \vee \neg x_4\}$. L'algorithme DPLL construit l'arbre suivant :



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 1.75. 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 1.76. Montrez que, pour tout $n \geq 1$, il existe un ensemble de clauses \mathcal{C}_n , dont les symboles propositionnels sont $\{p_1, \dots, p_n\}$ 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).

1.5.4.3 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, ils 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.

1.5.4.4 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 $O(1.3^n)$

1.5.5 Sous-classes de SAT

1.5.5.1 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 : <i>vrai</i> si \mathcal{C} est satisfaisable ou <i>faux</i> sinon

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

Algorithme 2-SAT :

Une clause d'ordre deux $l_1 \vee l_2$ est équivalente à $(\neg l_1 \Rightarrow l_2) \wedge (\neg l_2 \Rightarrow l_1)$. 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 $S = \{\neg p \mid p \in \text{PROP}(\varphi)\} \cup \text{PROP}(\varphi)$ (où $\text{PROP}(\varphi)$ est l'ensemble des variables propositionnelles de la formule φ). C'est l'ensemble des littéraux sur $\text{PROP}(\varphi)$
- l'ensemble des arcs est $A = \{(l_1, l_2) \mid \varphi \text{ contient une clause équivalente à } l_1 \Rightarrow l_2\}$.

Chaque clause $l_1 \vee l_2$ est donc associée à deux arcs $(\neg l_1, l_2)$ et $(\neg l_2, l_1)$.

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 \perp .

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. On rappelle qu'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 $O(|S| + |A|)$, donc 2-SAT est bien polynomial.

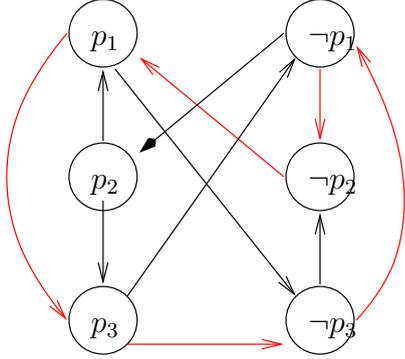
Exemple 1.77. $\varphi = (p_1 \vee p_2) \wedge (\neg p_1 \vee p_3) \wedge (\neg p_2 \vee p_1) \wedge (\neg p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_3)$

On a :

- $(p_1 \vee p_2) \equiv (\neg p_1 \Rightarrow p_2) \wedge (\neg p_2 \Rightarrow p_1)$

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

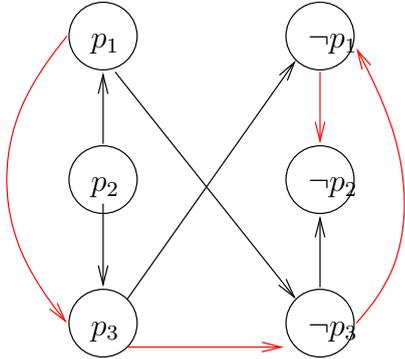
On construit le graphe $G(\varphi) = (S, A)$:



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 : $v(p_1) = v(p_2) = v(p_3) = 0$.

1.5.5.2 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 $\varphi_n = \ell_1 \vee \ell_2 \vee \dots \vee \ell_n$ avec $n \geq 3$ peut se mettre sous la forme :

$\psi_n = (\ell_1 \vee \ell_2 \vee q_1) \wedge (\ell_3 \vee \neg q_1 \vee q_2) \wedge \dots \wedge (\ell_{n-2} \vee \neg q_{n-4} \vee q_{n-3}) \wedge (\ell_{n-1} \vee \ell_n \vee \neg q_{n-3})$ 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.

1.5.5.3 Horn-SAT

Une *clause de Horn* est une clause comportant au plus un littéral positif. C'est donc une disjonction de la forme $\neg p_1 \vee \dots \vee \neg p_n \vee p$, 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) $\neg p_1 \vee \dots \vee \neg p_n \vee p$ avec $n > 0$;
- (b) $\neg p_1 \vee \dots \vee \neg p_n$ avec $n > 0$;
- (c) p ;
- (d) \perp ;

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 :

- $\neg p_1 \vee \dots \vee \neg p_n \vee p \equiv (p_1 \wedge p_2 \wedge \dots \wedge p_n) \Rightarrow p$ (avec $n > 0$) ;
- $\neg p_1 \vee \dots \vee \neg p_n \equiv (p_1 \wedge p_2 \wedge \dots \wedge p_n) \Rightarrow \perp$ avec $n > 0$;

Si q est une variable propositionnelle, et C est une clause de Horn, on note C/q la clause de Horn définie par

- $C/q = (p_2 \wedge \dots \wedge p_n) \Rightarrow p$ si $C = (p_1 \wedge p_2 \wedge \dots \wedge p_n) \Rightarrow p$, $p_1 = q$ et $n > 0$;
- $C/q = \text{vrai}$ si $C = (p_1 \wedge p_2 \wedge \dots \wedge p_n) \Rightarrow p$, $p = q$ et $n > 0$;
- $C/q = (p_2 \wedge \dots \wedge p_n)$ si $C = (p_1 \wedge p_2 \wedge \dots \wedge p_n)$, $p_1 = q$ et $n > 0$;
- $C/q = p$ si $C = p$ et $p \neq q$
- $C/q = \top$ si $C = q$
- $C/q = \perp$ si $C = \perp$

On étend cette notation à l'ensemble de clauses : $\mathcal{C}/q = \{C/q \mid C \in \mathcal{C}\}$.

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 satisfiable ?

$\mathcal{C} = \mathcal{C}_0$

tant que faits(\mathcal{C}) $\neq \emptyset$

Choisir $p \in$ faits(\mathcal{C})

$\mathcal{C} = \mathcal{C}/p$;

si $\perp \in \mathcal{C}$ **alors** Retourner "inconsistant"

sinon Retourner "satisfiable" ;

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

1.5.6 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 parmi ces nombreux solvers : zChaff datant de 2001 qui utilise l'algorithme 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 récentes de l'informatique, comme l'apprentissage, la programmation parallèle exploitant l'architecture multicoeur des processeurs récents, etc.

1.5.7 Applications

1.5.7.1 Vérification d'un circuit logique

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

1.5.7.2 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.

1.5.7.3 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 diagnostiquer 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 $Ab(S)$)

1.5.7.4 Planification

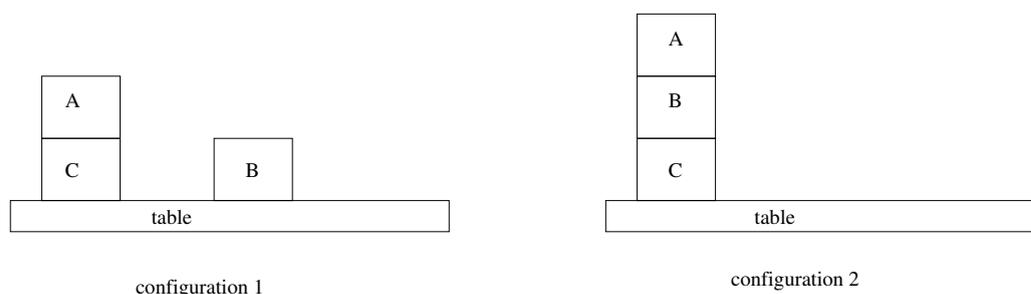
La planification est une discipline de l'intelligence artificielle qui vise le développement d'algorithmes pour produire des plans (en d'autres termes, une planification), typiquement pour l'exécution par un robot ou tout autre agent.

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 boîtes sur une table, 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 boîte non recouverte par une autre sur la table ou sur une autre boîte.



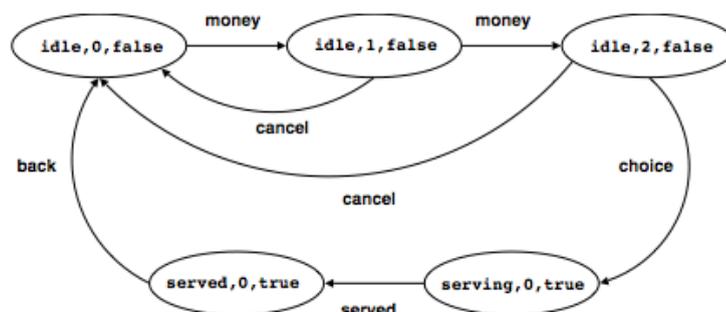
1.5.7.5 Model checking

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.

1.5.7.6 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).

1.5.7.7 Bio-informatique

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

1.5.8 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 importante 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 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.

1.6 Systèmes de preuves

1.6.1 La notion de système formel

Un *système de preuves* définit des règles d'inférence entre formules qui simulent le raisonnement. Il est défini par :

- le langage et les formules sur ce langage,
- les axiomes, c'est-à-dire les formules supposées être toujours vraies,
- les règles d'inférence. Une règle 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 grâce au système de preuves, i.e., s'il est possible de l'obtenir à partir des axiomes, en appliquant itérativement les règles d'inférence du système. Si Γ est un ensemble de formules, on note alors $\Gamma \vdash \varphi$ si φ peut se déduire dans le système de preuve 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 de preuve qui soit

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

Par extension, un *système formel* est un ensemble de règles permettant d'inférer des nouvelles conclusions à partir des prémisses. Les objets inférés peuvent être de nature différente des formules. Par exemple, le calcul des séquents (voir Section ??) est un système formel qui permet de déduire des couples de la forme (Γ, Δ) (notée comme habituellement par $\Gamma \vdash \Delta$), où Γ et Δ sont des multi-ensembles (i.e. des listes, mais où l'ordre ne compte pas) de formules, à partir d'autres couples du même type. Les théorèmes de correction et complétude pour le calcul des séquents montrent qu'on peut déduire le couple (Γ, Δ) dans le calcul si et seulement si la formule $\bigvee_{\delta \in \Delta} \delta$ est une conséquence logique de Γ .

1.6.2 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 2.7). 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.

1.6.2.1 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 1.4) :

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).

$\frac{C \vee \ell \vee \ell}{C \vee \ell} \text{ factorisation}$	
$\frac{C \vee \ell \quad C' \vee \neg \ell}{C \vee C'} \text{ coupure}$	$\frac{\ell \quad \neg \ell}{\perp} \text{ coupure (cas particulier)}$

FIGURE 1.4 – Règles d'inférence pour la méthode de la coupure

On peut utiliser ce système formel pour :

1. montrer qu'une formule (ou un ensemble de formules) est contradictoire (ou admet un modèle);
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 formule φ , on suppose $\neg\varphi$ et on montre que cela conduit à une contradiction;
3. montrer qu'une formule φ est une conséquence logique d'un ensemble de formules Γ : on montre que l'ensemble $\Gamma \cup \{\neg\varphi\}$ est contradictoire.

Exemple 1.78. Soit la formule $\varphi = (p \vee r \vee s) \wedge (r \vee \neg s) \wedge \neg r \wedge \neg p$; cette formule s'écrit comme l'ensemble de clauses $\mathcal{C} = \{(p \vee r \vee s), (r \vee \neg s), \neg r, \neg p\}$. Voici une dérivation par résolution à partir de cet ensemble de clauses :

$$\frac{\frac{\frac{p \vee r \vee s \quad r \vee \neg s}{\text{coupure}}}{p \vee r \vee r}{\text{factorisation}}}{p \vee r} \quad \frac{\neg r}{\text{coupure}}}{\frac{p}{\text{coupure}} \quad \frac{\neg p}{\text{coupure}}} \perp$$

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

Exemple 1.79. L'exemple suivant montre qu'on peut avoir besoin d'utiliser deux fois une clause pour dériver la clause vide. Soit φ la formule $((p \Rightarrow q) \wedge (q \Rightarrow p) \wedge (p \vee q)) \Rightarrow (p \wedge q)$. On veut montrer que φ est une tautologie, on met donc $\neg\varphi$ sous forme clausale, on obtient l'ensemble de clause $\mathcal{C} = \{(p \vee \neg q), (\neg p \vee q), (p \vee q), (\neg p \vee \neg q)\}$. Une preuve par la méthode de la coupure est la suivante :

$$\frac{\frac{\frac{p \vee \neg q \quad \neg q \vee \neg p}{\text{coupure}}}{\neg q \vee \neg q}{\text{factorisation}}}{\neg q} \quad \frac{\neg p \vee q}{\text{coupure}}}{\frac{\neg p}{\text{coupure}} \quad \frac{p \vee q}{\text{coupure}}} \quad \frac{q}{\text{coupure}} \quad \frac{\neg q}{\text{coupure}}}{\perp}$$

Donc nous avons montré qu'on peut dériver \perp à partir de $\mathcal{C} = \{(p \vee \neg q), (\neg p \vee q), (p \vee q), (\neg p \vee \neg q)\}$.

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 peut donc jamais dériver la clause vide.

Remarque 1.80. L'exemple précédent montre que l'ordre de littéraux dans une clause n'est pas important quand on applique la règle de coupure. Nous avons en fait inféré comme suit :

$$\frac{p \vee \neg q \quad \neg q \vee \neg p}{\neg q \vee \neg q} \text{ coupure}$$

au lieu d'avoir, dans la première prémisse, $\neg q \vee p$.

De façon semblable, l'ordre de littéraux n'est pas important afin d'appliquer la règle de factorisation.

1.6.2.2 Correction et complétude

Notation 1.81. 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. C'est-à-dire, $\mathcal{C} \vdash_R \psi$ signifie qu'il existe une suite finie d'ensembles de clauses $\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_n$ vérifiant :

1. $\mathcal{C}_0 = \mathcal{C}$;

2. pour tout $i \in [0, n-1]$, $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{\psi_{i+1}\}$, où ψ_{i+1} est la conséquence d'une règle du système (factorisation ou coupure) dont les prémisses appartiennent à \mathcal{C}_i ;
3. $\mathcal{C}_n = \mathcal{C}_{n-1} \cup \{\psi_n\}$, avec $\psi_n = \perp$.

Théorème 1.82. *La résolution est correcte ; c'est-à-dire, si $\mathcal{C} \vdash_R \perp$, alors \mathcal{C} est insatisfaisable.*

Démonstration. $\mathcal{C} \vdash_R \perp$ signifie qu'il existe une suite finie d'ensembles de clauses $\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_n$ 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 coupure à \mathcal{C}_i ;
- \mathcal{C}_n contient la clause \perp .

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 résolvente 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 1.35 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 \perp , il est donc est insatisfaisable, donc $\mathcal{C} = \mathcal{C}_0$ est insatisfaisable. \square

Définition 1.83. Un ensemble de clauses est dit *saturé* si on ne peut pas produire des nouvelles clauses par application de la règle de coupure.

Théorème 1.84. *La résolution est complète. C'est-à-dire : si \mathcal{C} est insatisfaisable, alors $\mathcal{C} \vdash_R \perp$.*

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

Nous allons montrer que si $\mathcal{C} \not\vdash_R \perp$, alors \mathcal{C} est satisfaisable. Or la condition $\mathcal{C} \not\vdash_R \perp$ revient à dire que $\perp \notin \mathcal{S}$, où $\mathcal{S} = \{C \mid \mathcal{C} \vdash_R C\}$. Il est facile à voir que \mathcal{S} est un ensemble saturé ; même si potentiellement infini, l'ensemble des variables propositionnelles qui ont une occurrence dans \mathcal{S} est fini. Nous allons donc montrer qu'un tel ensemble \mathcal{S}

- saturé,
- avec un nombre fini de variables propositionnelles,
- tel que $\perp \notin \mathcal{S}$,

est satisfaisable ; le résultat découle ensuite du fait que $\mathcal{C} \subseteq \mathcal{S}$ et donc $\text{mod}(\mathcal{S}) \subseteq \text{mod}(\mathcal{C})$.

On suppose donc que \mathcal{S} est saturé et ne contient pas la clause \perp 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 : $n = 1$. Soit donc p le seul symbole propositionnel qui a une occurrence dans \mathcal{S} .

Car il est saturé, \mathcal{S} ne peut contenir à la fois les deux clauses unitaires p et $\neg p$, sinon il contiendrait aussi 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 :

$$\begin{aligned} \mathcal{S}_0 &:= \{C \in \mathcal{S} \mid \neg p \notin C\}, & \mathcal{S}'_0 &:= \mathcal{S}_0[p \leftarrow \perp], \\ \mathcal{S}_1 &:= \{C \in \mathcal{S} \mid p \notin C\}, & \mathcal{S}'_1 &:= \mathcal{S}_1[p \leftarrow \top]. \end{aligned}$$

En utilisant la notation introduite avec la Remarque 1.71, on a $\mathcal{S}'_0 = \mathcal{S}[p \leftarrow \perp]$ et $\mathcal{S}'_1 = \mathcal{S}[p \leftarrow \top]$. Remarquez que $\perp \in \mathcal{S}'_0$ implique que $p \in \mathcal{S}_0$; aussi \mathcal{S}'_0 est saturé. Par exemple, pour $q \neq p$, on a bien

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

$$\frac{C_1 \vee q \quad C_2 \vee \neg q}{C_1 \vee C_2} \rightsquigarrow \frac{C_1[p \leftarrow \perp] \vee q \quad C_2[p \leftarrow \perp] \vee \neg q}{C_1[p \leftarrow \perp] \vee C_2[p \leftarrow \perp]}$$

FIGURE 1.5 – Commutation de la coupure par rapport à la substitution

comme suggéré par la Figure 1.6.2.2. De même, \mathcal{S}'_1 est saturé, et $\perp \in \mathcal{S}'_1$ implique $\neg p \in \mathcal{S}_1$. En particulier, si $\perp \in \mathcal{S}'_0 \cap \mathcal{S}'_1$, 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 $\perp \in \mathcal{S}$, une contradiction.

Par conséquent, un parmi \mathcal{S}'_i , $i = 0, 1$, est saturé 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, soit v un modèle de \mathcal{S}'_1 . Soit u la valuation telle que $u(p) = 1$ et $u(q) = v(q)$ pour $p \neq q$; alors u est un modèle de \mathcal{S} . En fait, si $C \in \mathcal{S}_1$, alors $p \notin C$, et

$$1 = v(C[p \leftarrow \top]) = u(C[p \leftarrow \top]) \leq u(C \vee \neg p) = \max(u(C), 0) = u(C);$$

sinon $C \notin \mathcal{S}_1$, donc $p \in C$, et $1 = u(p) \leq u(C)$. \square

1.6.2.3 L'algorithme de résolution

On peut extraire, de la méthode de la coupure, un (semi)-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, $\neg\varphi = C_1 \wedge \dots \wedge C_n$, où toute clause est factorisée;
2. remplacer la conjonction de clauses $C_1 \wedge \dots \wedge C_n$ par l'ensemble $\mathcal{C} = \{C_1, \dots, C_n\}$;
3. saturer l'ensemble en rajoutant à \mathcal{C} les clauses (factorisés) qu'on peut déduire à partir des deux règles de résolution.

L'algorithme termine lorsque :

- (a) soit on vient d'ajouter la clause \perp et donc la formule initiale est insatisfaisable;
- (b) soit l'application des règles ne modifie plus l'ensemble de clauses.

Malheureusement, seulement (a) — l'ensemble des clauses contient \perp — peut être considéré comme un vrai critère de terminaison. 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 résolventes qui ne sont pas déjà dans l'ensemble de clauses, soit avoir un critère permettant d'arrêter de générer de nouvelles résolventes.

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 propriétés de complétude et correction).

Exemple 1.85. 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.

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

Exemple 1.86. $\mathcal{C} = \{p \vee \neg p \vee \neg q\}$. On commence par appliquer la règle de coupure, 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 coupure aux clauses $p \vee \neg p \vee \neg q$ et $p \vee \neg p \vee \neg q \vee \neg q$.

$$\frac{\frac{p \vee \neg p \vee \neg q \quad p \vee \neg p \vee \neg q}{p \vee \neg p \vee \neg q \vee \neg q} \text{ coupure} \quad p \vee \neg p \vee \neg q}{\frac{p \vee \neg p \vee \neg q \vee \neg q \vee \neg q}{p \vee \neg p \vee \neg q} \text{ coupure}} \text{ coupure}$$

$$\vdots$$

La méthode va générer les clauses de la forme $p \vee \neg p \vee \neg q \vee \neg q \vee \dots \neg q$ 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 le résultat démontré dans la preuve de complétude : un ensemble de clauses saturé qui ne contient pas la clause vide a un modèle. Donc la formule $\neg(p \vee \neg p \vee \neg q)$ n'est pas une tautologie.

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

Définition 1.87. Une 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. L'algorithme se présente donc comme en Figure 1.6. La terminaison de l'algorithme est assurée par le fait que le nombre de clauses factorisées ayant au plus n symboles propositionnels est borné supérieurement par une fonction de n .

Algorithme Res
entrée : un ensemble de clauses \mathcal{C} factorisés
sortie : <i>insatisfaisable</i> si $\mathcal{C} \models \perp$ ou <i>satisfaisable</i> sinon
si $\perp \in \mathcal{C}$, alors retourner : <i>insatisfaisable</i> ; (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 : <i>satisfaisable</i> .

FIGURE 1.6 – L'algorithme de coupure (ou de résolution)

Exercice 1.88. Estimer cette borne supérieure.

L'algorithme peut aussi être optimisé si on maintient 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 . 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 $\text{mod}(\mathcal{C}) = \text{mod}(\mathcal{C} \setminus \{C_1\})$.

1.7 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 quatre notions recouvrent les formes suivantes :

$$\begin{array}{ll} \text{Clause conjonctive : } & \bigwedge_{i=1}^n \ell_i ; & \text{Clause disjonctive : } & \bigvee_{i=1}^n \ell_i ; \\ \text{FNC : } & \bigwedge_{j=1}^m \bigvee_{i=1}^n \ell_i^j ; & \text{FND : } & \bigvee_{j=1}^m \bigwedge_{i=1}^n \ell_i^j . \end{array}$$

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 $\text{Val}(\text{PROP})$ (ou juste Val lorsqu'il n'y a pas d'ambiguïté 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 $\text{mod}(\varphi)$ 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 $\text{mod}(\varphi) = \text{mod}(\psi)$, 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 $\text{mod}(\Gamma)$ 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 $\text{mod}(\Gamma) \subseteq \text{mod}(\varphi)$. 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 $\text{mod}(\Gamma) \subseteq \text{mod}(\Sigma)$.

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

- [Sil99] João P. Marques Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence : Progress in Artificial Intelligence*, EPIA '99, pages 62–74, London, UK, UK, 1999. Springer-Verlag.

Chapitre 2

Calcul des prédicats

2.1 Introduction

Le calcul des propositions est bien trop limité pour décrire des situations réelles. En effet il ne permet que de décrire des phrases dont la vérité ne dépend pas des individus (par exemple « Il pleut ») ; il ne peut pas représenter des phrases qui mettent en jeu des individus ou des objets (par exemple « Si x est le père de y et si z est le père de x alors z est un grand-père de y » ou « Tout individu a un père »).

Le calcul des prédicats (ou Logique du Premier Ordre) permet d'exprimer de telles relations entre individus, il est donc bien plus riche que le calcul propositionnel. En premier lieu, il contient des individus (ou entités) (donnés par des symboles de variables x, y, z, \dots). Il contient des fonctions (f, g, \dots, s, \dots) permettant de transformer des entités en autres entités (par exemple la fonction qui associe une personne à son père), et des relations (\dots, P, Q, R, \dots) permettant de lier les individus entre eux.

Les relations appliquées aux entités (par exemple $R(x, f(y))$) peuvent être évaluées à vrai ou faux (selon les valeurs attribuées aux entités, aux fonctions et aux relations) et servent de briques de base à un langage du premier ordre obtenu à l'aide des connecteurs logiques du calcul propositionnel et de deux autres connecteurs appelés quantificateurs.

Le calcul des prédicats est donc très similaire à celui des propositions. On aura des formules définies inductivement à partir des symboles de prédicats et de fonctions. On les interprétera dans divers mondes possibles et alors elles deviendront vraies ou fausses. On aura également un système formel correct et complet pour démontrer ou réfuter des formules.

Il y a néanmoins une différence algorithmique importante : le calcul des prédicat est indécidable : il est absolument impossible de vérifier qu'une formule est vraie pour toute interprétation. Ceci vient du fait que :

1. il existe un nombre non borné d'interprétations possible. Il n'est donc pas possible de toutes les tester une à une
2. les interprétations comportent en général une infinité d'individus, il est donc pas toujours possible de vérifier qu'une interprétation donnée est modèle d'une formule (par exemple de vérifier que $\forall x \varphi$ puisque x peut prendre une infinité de valeurs différentes).

2.2 Préliminaires - Exemple introductif

Nous rappelons ici les notions de base sur les relations et fonctions.

2.2.1 Produit d'un ensemble

Si D est un ensemble, alors D^n dénotera le produit cartésien de D avec lui-même n -fois. C'est l'ensemble des tuplets de longueur n dont les éléments sont tous tirés de D . Soit :

$$D^n = \underbrace{D \times D \times \dots \times D}_{n\text{-fois}} := \{(d_1, d_2, \dots, d_n) \mid d_i \in D, \text{ pour } i = 1, \dots, n\}.$$

Remarquez que nous pouvons donner un sens à l'ensemble D^n avec $n = 0$:

$$D^0 := \{()\}.$$

Notez que D^0 ne désigne pas l'ensemble vide !

2.2.2 Les relations

Etant donné un ensemble E , et $n \geq 0$ un entier. Une relation n -aire (ou d'arité n) sur E est un sous-ensemble de E^n .

Exemple 2.1.

1. $E = \{1, 2, 3\}$ et R est la relation binaire (d'arité 2) définie par $R = \{(1, 1), (2, 2), (3, 3)\}$.
2. $E = \mathbb{N}$ et S est la relation d'arité 2 définie par $S = \{(n, n + 1) \mid n \in \mathbb{N}\}$.
3. $E = \{1, 2, 3\}$ et R est la relation unaire (d'arité 1) définie par $R = \{1, 2\}$.

Si R est une relation d'arité n , on note $R(a_1, \dots, a_n)$ ssi $(a_1, \dots, a_n) \in R$.

2.2.3 Les fonctions

Etant donné un ensemble E , et n un entier positif, une fonction n -aire (ou d'arité n) sur E est une fonction de E^n dans E . Une fonction n'est pas forcément une application : elle peut être non définie pour certains éléments de E^n , dans ce cas on dira que c'est une fonction partielle.

Exemple 2.2.

1. $E = \{1, 2, 3\}$ et f est la fonction binaire (d'arité 2) définie pour tout couple $(a, b) \in E^2$ par :
 - $f(a, b) = 1$ si $a = 1$ et $b = 2$,
 - $f(a, b) = 2$ si $a = 2$ et $b = 3$,
 - $f(a, b) = 3$ si $a = 3$ et $b = 1$,
 - $f(a, b)$ est indéfinie sinon (i.e., pour les couples $(1, 1), (2, 2), (3, 3), (3, 2), (2, 1), (1, 3)$).
2. $E = \mathbb{N}$ et f est la fonction d'arité 1 définie pour tout $n \in \mathbb{N}$ par $f(n) = n + 1$.

Remarque 2.3. En particulier, le lecteur observera que les fonctions d'arité 0, $f : D^0 \rightarrow D$ sont en bijection avec les éléments $d \in D$ (via l'évaluation $f() = d \in D$). Nous allons donc identifier une fonction d'arité 0 à une constante, c'est-à-dire à un élément $d \in D$.

2.2.4 Un exemple

Avant d'en venir aux définitions formelles, voici quelques formules du premier ordre. Ici, P et Q désignent deux relations binaires et f une fonction unaire. Les lettres x, y et z désignent des variables. Vous pouvez reconnaître également les connecteurs logique introduits dans le calcul propositionnel, ainsi que deux quantificateurs que vous connaissez certainement \forall qui se lit « pour tout » et \exists qui se lit « il existe ».

$$\varphi_G : \forall x \forall y \forall z (P(x, y) \wedge P(y, z)) \Rightarrow G(x, z)$$

$$\varphi_P : \forall x \exists y P(y, x)$$

$$\varphi_C : \forall x \exists y G(y, x)$$

$$\varphi_D : \forall x \forall z P(z, f(x)) \Rightarrow G(z, x)$$

$$\varphi_F : (\varphi_G \wedge \varphi_P) \Rightarrow \varphi_C.$$

Il est ici impossible de donner une valeur de vérité à toutes ces formules, et ce pour différentes raisons :

- on ne sait pas dans quel ensemble D sont prises les valeurs x, y, z
- on ne connaît pas la valeur de la fonction $f : D \rightarrow D$
- on ne connaît pas la valeur des relations $P \subseteq D \times D$, et $G \subseteq D \times D$

Il est donc nécessaire de choisir une interprétation pour évaluer ces formules. Toutefois, nous verrons que c'est inutile pour la formule φ_F qui est vraie pour toute interprétation (c'est une tautologie, ou un théorème).

Considérons donc diverses interprétations de ces formules.

2.2.4.1 Interprétation 1

Les individus sont les êtres humains. La relation $P(x, y)$ signifie que x est le père de y . La relation $G(x, y)$ signifie que x est un grand-père de y . La fonction f associe à un individu sa mère.

φ_G signifie alors : pour tous êtres humains x, y, z , si (x est le père de y et y est le père de z) alors (x est un grand-père de z).

φ_P signifie : « pour tout individu x il existe un individu y tel que y est le père de x » soit, plus simplement, « tout individu a un père ».

φ_C signifie que « pour tout individu x il existe un individu y tel que y est le grand-père de x » soit, plus simplement, « tout individu a un grand-père ».

φ_D dit que si z est le père de la mère de x alors z est un grand père de x .

Ces quatre formules sont vraies dans cette interprétation ¹.

L'implication $\varphi_F : ((\varphi_G \wedge \varphi_P) \Rightarrow \varphi_C)$ est donc vraie dans cette interprétation.

On remarquera que les deux formules φ_P et φ_G sont loin de modéliser toutes les propriétés des relations G et P . On n'a pas dit que « le père de chaque individu est unique », ni qu'« un individu x peut-être le grand-père d'un autre z sans qu'il existe un individu dont x soit le père et qui soit le père de z » (le grand-père maternel).

2.2.4.2 Interprétation 2

Les individus sont les quatre sommets a, b, c, d du graphe suivant :

$$\begin{array}{ccc} a & \rightarrow & b \\ \uparrow & & \downarrow \\ d & \leftarrow & c \end{array}$$

$P(x, y)$ signifie que x précède immédiatement y sur le graphe.

$G(x, y)$ signifie que x suit immédiatement y sur le graphe.

La formule φ_P signifie que tout point a un prédécesseur immédiat, et elle vraie.

La formule φ_G signifie que pour tous points x, y, z , si x précède immédiatement y et si y précède immédiatement z , alors x suit immédiatement z . Elle est fautive. Finalement la formule φ_C signifie que tout point a un successeur immédiat. Elle est également vraie.

L'implication $((\varphi_P \wedge \varphi_G) \Rightarrow \varphi_C)$ est donc vraie dans cette interprétation.

2.2.4.3 Interprétation 3

Les individus sont les nombres entiers positifs.

$P(x, y)$ signifie que $y = x + 1$. Par exemple $P(4, 5)$ est vraie, mais $P(5, 4)$ est fautive.

$G(x, y)$ signifie que $y = x + 2$. Par exemple $P(4, 6)$ est vraie, mais $P(6, 4)$ est fautive.

La formule φ_P signifie alors que pour tout entier y , il existe un entier x tel que $y = x + 1$. Elle est fautive : pour $y = 0$ un tel entier positif n'existe pas.

La formule φ_G signifie que pour tous entiers x, y, z , si $y = x + 1$ et $z = y + 1$ alors $z = x + 2$. Elle est vraie.

Finalement la formule φ_C signifie que pour tout entier x , il existe un entier z tel que $x = z + 2$. Elle est fautive : pour $x = 0$ il n'existe pas de tel entier positif.

L'implication $((\varphi_P \wedge \varphi_G) \Rightarrow \varphi_C)$ est donc vraie dans cette interprétation.

1. Sauf peut-être φ_P : soit il y a un premier homme et celui-ci n'a pas de père, soit on se retrouve peu à peu, en remontant l'évolution, à inclure dans le genre humains des singes, des poissons, des bactéries, ...

2.2.4.4 Comparaison des interprétations

On est donc tout à fait libre d'interpréter les formules dans un « monde » de son choix, de sorte que certains énoncés deviennent vrais ou faux. On remarque néanmoins que pour chacune des interprétations considérées, l'implication $((\varphi_P \wedge \varphi_G) \Rightarrow \varphi_C)$ est vraie. Ce n'est pas un hasard, cette formule est une tautologie du calcul des prédicats. Elle est vraie dans toute interprétation.

2.3 Expressions et formules

2.3.1 Les termes

Définition 2.4 (Signature). Une *signature* est un ensemble \mathcal{S} de symboles muni d'une application $\rho : \mathcal{S} \rightarrow \mathbb{N}$, appelée *arité*.

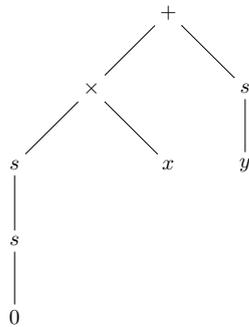
On écrira une signature comme un ensemble de couples. Par exemple, $\{(f, 2), (g, 1), (h, 0)\}$ est la signature dont les symboles sont f, g, h , d'arité 2, 1 et 0, respectivement. Formellement, on a ici $\mathcal{S} = \{f, g, h\}$, $\rho(f) = 2$, $\rho(g) = 1$, $\rho(h) = 0$. Un symbole $f \in \mathcal{S}$ tel que $\rho(f) = 0$ est appelé *constante*.

Définition 2.5 (Termes). Étant donné une signature \mathcal{S} et un ensemble X (de variables individuelles), l'ensemble $\mathcal{T}_{\mathcal{S}}(X)$ des termes est le plus petit ensemble tel que :

- $x \in \mathcal{T}_{\mathcal{S}}(X)$ pour toute variable $x \in X$
- si $f \in \mathcal{S}$ est d'arité $n \geq 0$ et si $t_1, \dots, t_n \in \mathcal{T}_{\mathcal{S}}(X)$, alors $f(t_1, \dots, t_n) \in \mathcal{T}_{\mathcal{S}}(X)$.

C'est-à-dire qu'un terme est une expression formée à partir de X en utilisant les symboles de \mathcal{S} de sorte qu'un symbole f soit appliqué à un nombre de termes égal à $\rho(f)$. En particulier, si $f \in \mathcal{S}$ est une constante, alors elle s'applique à une liste vide de termes : $f()$ est un terme ; pour simplifier la notation, on écrit également f . Un terme est dit **clos** si il ne contient aucune variable.

Exemple 2.6. La signature de l'arithmétique contient la constante 0, le symbole s d'arité 1 (qui représente la fonction « successeur »), et les symboles $+$ et \times d'arité 2. On emploie la notation $\mathcal{S} = \{(0, 0), (s, 1), (+, 2), (\times, 2)\}$ pour représenter cette signature. Ainsi, pour $x, y \in X$, l'expression $+(\times(s(s(0)), x), s(y))$ (que nous nous autoriserons à écrire $(s(s(0)) \times x) + s(y)$) est un terme de $\mathcal{T}_{\mathcal{S}}(X)$ qu'on peut représenter par l'arbre suivant :



En fait, tout terme est un arbre à branchements finis. Leur hauteur (et donc leur taille) n'est par contre pas bornée car, par exemple,

$$s^n(0) := \underbrace{s(s(\dots s(0)\dots))}_{n\text{-fois}}$$

est un terme, pour tout $n \geq 1$.

Exemple 2.7. Le signature de la théorie des groupes est $\{(e, 0), (inv, 1), (*, 2)\}$, où $*$ est l'opérateur de composition et inv est l'opérateur « inverse » qui est habituellement noté x^{-1} .

2.3.2 Le langage

Définition 2.8 (Langage). Un *langage* (ou *vocabulaire*) du premier ordre est la donnée d'un couple de signatures $\mathcal{S} = (\mathcal{S}_f, \mathcal{S}_r)$ disjointes (i.e. avec $\mathcal{S}_f \cap \mathcal{S}_r = \emptyset$).

On dit que :

- Les éléments de \mathcal{S}_f sont les *symboles de fonction* du langage.
- Les éléments de \mathcal{S}_r sont les *symboles de relation* (ou de prédicat) du langage.

Remarque 2.9.

1. Rappelons qu'une constante est un symbole (de fonction) d'arité 0.
2. On considérera (sauf mention contraire) que chaque langage contient le symbole de relation binaire $=$ et un symbole de relation d'arité 0, \perp qui représentera le faux. Un langage contenant le symbole binaire $=$ sera appelé **langage égalitaire**.
3. Le rôle des relations et des fonctions est très différent. Les fonctions et constantes seront utilisés pour construire les termes (i.e., les objets du langage) tandis que les relations serviront à construire des formules (i.e., des propriétés sur ces objets)
Par exemple $1 + 2$ est un terme, il désigne un objet, tandis que $1 + 2 = 3$ désigne une formule logique.

Exemple 2.10.

1. Le langage \mathcal{L}_1 de la théorie des groupes contient les symboles
 - de constante : e ;
 - de fonction : $*$ (binaire) et inv (unaire);
 - de relation : $=$ (binaire).
2. Le langage \mathcal{L}_2 de la théorie des ensembles contient les symboles
 - de constante : \emptyset ;
 - de fonction : \cap et \cup binaires, C unaire (le complément);
 - de relation : $=$, \in et \subset , tous binaires.²

2.3.3 Les formules du calcul des prédicats

Etant donné un langage $\mathcal{S} = (\mathcal{S}_f, \mathcal{S}_r)$, on construit les formules de la logique du premier ordre en utilisant les connecteurs de la logique propositionnelle et deux quantificateurs : \forall et \exists .

Les formules sont construites à partir des formules atomiques, qui sont elles mêmes construites à partir des termes. Une formule atomique est obtenue en appliquant un symbole de relation à des termes.

On se fixe pour toute la suite un ensemble $X = \{x, x_0, x_1, \dots, y, y_0, y_1, \dots, z, z_0, z_1, \dots\}$ de variables.

Définition 2.11 (Formules atomiques). Soit $\mathcal{S} = (\mathcal{S}_f, \mathcal{S}_r)$ un langage, une formule atomique sur \mathcal{S} est une expression de la forme suivante :

- $r(t_1, \dots, t_n)$ où $r \in \mathcal{S}_r$ est d'arité $n \geq 0$ et $t_1, t_2, \dots, t_n \in \mathcal{T}_{\mathcal{S}_f}(X)$ sont des termes.

Définition 2.12 (Formules). Etant donné un langage \mathcal{S} , l'ensemble $\mathcal{F}_{po}(\mathcal{S})$ des formules du premier ordre sur \mathcal{S} est le plus petit ensemble tel que :

1. toute formule atomique sur \mathcal{S} appartient à $\mathcal{F}_{po}(\mathcal{S})$,
2. si $\varphi, \psi \in \mathcal{F}_{po}(\mathcal{S})$ sont deux formules alors :
 - $\varphi \wedge \psi \in \mathcal{F}_{po}(\mathcal{S})$,
 - $\varphi \vee \psi \in \mathcal{F}_{po}(\mathcal{S})$,
 - $\varphi \Rightarrow \psi \in \mathcal{F}_{po}(\mathcal{S})$,
 - $\neg\varphi \in \mathcal{F}_{po}(\mathcal{S})$,

² En fait, en théorie des ensembles, seulement les symboles $=$ et \in sont considérés élémentaires. Les autres se définissent à partir de ces deux.

3. si $\varphi \in \mathcal{F}_{\text{po}}$ et $x \in X$ alors
- $\forall x\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$,
 - $\exists x\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$.

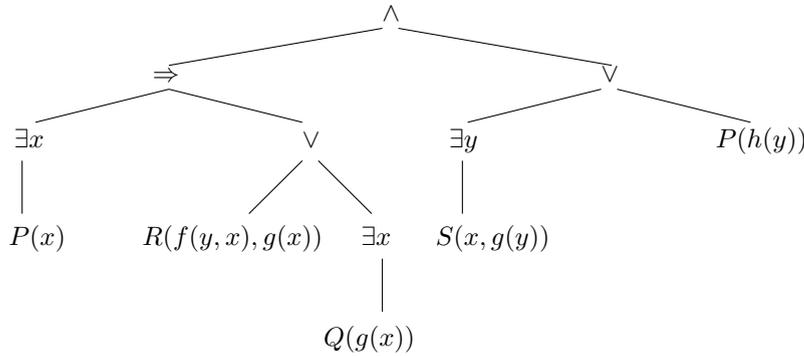
Exemple 2.13. Reprenons les langages de l'exemple 2.10 :

1. Dans \mathcal{L}_1 , $x * y = e$, $y * x = e$, $x = e$, $y = e$ et $y * x = e$ sont des formules atomiques. Par conséquent, $\forall x\exists y((x * y = e) \wedge (\neg(y * x = e)))$ et $\forall x((x = e) \vee \exists y((\neg(y = e)) \wedge (y + x = e)))$ sont des formules.
2. Dans \mathcal{L}_2 , $\forall x\forall y(x \cap y = \emptyset \Rightarrow (x = \emptyset \wedge y = \emptyset))$ est une formule.

Remarquez que, comme pour les termes, les formules peuvent être vues comme des arbres dont les feuilles sont des formules atomiques et les noeuds sont les connecteurs et quantificateurs. Par exemple, la formule

$$[(\exists xP(x)) \Rightarrow (R(f(y, x), g(x)) \vee [\exists xQ(g(x))])] \wedge [(\exists yS(x, g(y))) \vee P(h(y))] \quad (2.1)$$

sera représentée **de façon unique** par l'arbre suivant :



2.3.4 Occurrences libres et liées d'une variable

Lorsqu'une variable x appartient à une sous-formule précédée d'un quantificateur, $\forall x$ ou $\exists x$, elle est dite **liée** par ce quantificateur. Si une variable n'est liée par aucun quantificateur, elle est **libre**.

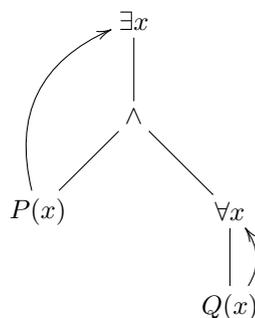
La distinction entre variable libre et variable liée est importante. Une variable liée ne possède pas d'identité propre et peut être remplacée par n'importe quel autre nom de variable qui n'apparaît pas dans la formule. Ainsi, $\exists x(x < y)$ est identique à $\exists z(z < y)$ mais pas à $\exists x(x < z)$ et encore moins à $\exists y(y < y)$.

L'ensemble des variables libres $\text{FV}(\varphi)$ (depuis l'anglais, *free variable*), et l'ensemble des variables liées $\text{BV}(\varphi)$ (depuis l'anglais, *bound variable*)³ d'une formule φ sont définis par induction sur la structure de φ :

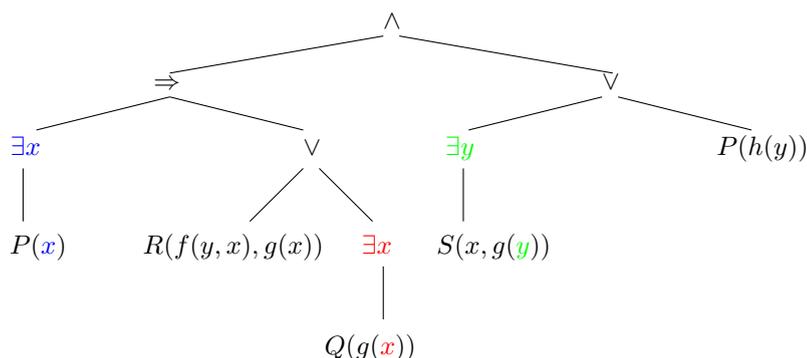
- si φ est une formule atomique, alors toute occurrence d'une variable x dans φ est libre : $\text{FV}(\varphi) = \text{Var}(\varphi)$ et $\text{BV}(\varphi) = \emptyset$.
- si φ est $\exists x\psi$ ou $\forall x\psi$, alors $\text{FV}(\varphi) = \text{FV}(\psi) - \{x\}$; $\text{BV}(\varphi) = \text{BV}(\psi) \cup \{x\}$ (et toute occurrence de x **libre** dans ψ devient liée dans φ par le quantificateur introduit);
- si $\varphi = \varphi_1 \circ \varphi_2$ (où $\circ \in \{\wedge, \vee, \Rightarrow\}$) alors $\text{FV}(\varphi) = \text{FV}(\varphi_1) \cup \text{FV}(\varphi_2)$, $\text{BV}(\varphi) = \text{BV}(\varphi_1) \cup \text{BV}(\varphi_2)$;
- si $\varphi = \neg\psi$, alors $\text{FV}(\varphi) = \text{FV}(\psi)$ et $\text{BV}(\varphi) = \text{BV}(\psi)$.

Définition 2.14 (Formule close). Une formule φ est dite **close** ssi $\text{FV}(\varphi) = \emptyset$.

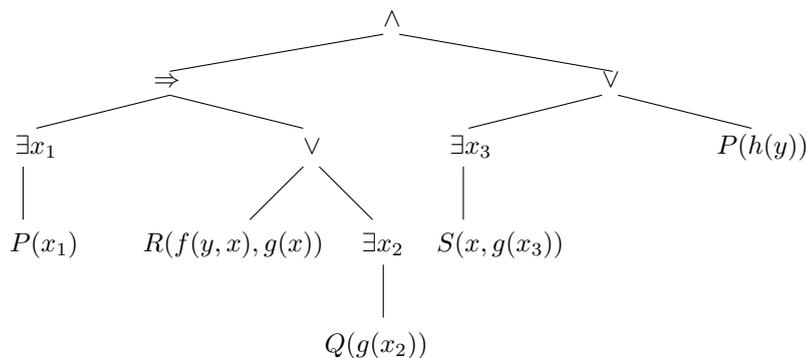
Exemple 2.15. Une variable liée est attachée à une et une seule occurrence d'un quantificateur dans la formule : celui qui dans l'arbre est son ancêtre le plus proche. Par exemple, considérons la formule $\exists xP(x) \wedge \forall xQ(x)$. Les variables sont liées de la façon suivante :



Exemple 2.16. Considérons la formule (2.1) et son arbre :



Une variable est liée si elle a une occurrence dans une feuille qui est descendant d'un noeud étiqueté par un quantificateur avec cette variable. Depuis l'exemple précédent, nous observons qu'il peut bien être le cas que $FV(\varphi) \cap BV(\varphi) \neq \emptyset$. Nous verrons par la suite que les variables liées peuvent être renommée sans modifier la sémantique d'une formule, ceci par exemple pour éviter les ambiguïtés. Ainsi, nous pourrons toujours supposer que $FV(\varphi) \cap BV(\varphi) = \emptyset$. Par exemple, la formule précédente peut-être renommée de la façon suivante :



La nouvelle formule est donc

$$([\exists x_1 P(x_1)] \Rightarrow [(\exists x_2 Q(g(x_2))) \vee R(f(y, x), g(x))]) \wedge [(\exists x_3 S(x, g(x_3))) \vee P(h(y))] .$$

2.4 Sémantique

Jusqu'ici nous nous sommes contentés de définir la **syntaxe** des langages. Les formules n'ont encore aucune signification, en partie car nous n'avons pas donné de signification aux symboles des langages. Une signature ne donne qu'un ensemble de symboles, sans en donner d'interprétation.

2.4.1 Structures

Pour donner une **sémantique** aux formules, il faut donc commencer par donner une signification aux éléments de la signature du langage. On fait cela en associant une *structure* au langage.

Définition 2.17. Soit $\mathcal{S} = (\mathcal{S}_r, \mathcal{S}_f)$ un langage. Une **\mathcal{S} -structure** (ou **\mathcal{S} -interprétation**) \mathcal{M} est la donnée d'un ensemble $D_{\mathcal{M}}$ et

- (a) d'une relation $R^{\mathcal{M}} \subseteq D_{\mathcal{M}}^{\rho(R)}$, pour chaque symbole de relation $R \in \mathcal{S}_r$;
- (b) d'une fonction totale (application) $f^{\mathcal{M}} : D_{\mathcal{M}}^{\rho(f)} \rightarrow D_{\mathcal{M}}$, pour chaque symbole de fonction $f \in \mathcal{S}_f$.

Remarque 2.18. A cause de la remarque 2.3, la clause (b) dans la définition d'une \mathcal{S} -structure peut se rephraser comme suit :

- (b.1) d'un élément $c^{\mathcal{M}}$ de $D_{\mathcal{M}}$, pour chaque symbole de constante (symbole de fonction d'arité 0) $c \in \mathcal{S}_f$,
- (b.2) d'une fonction totale (application) $f^{\mathcal{M}} : D_{\mathcal{M}}^{\rho(f)} \rightarrow D_{\mathcal{M}}$, pour chaque symbole de fonction $f \in \mathcal{S}_f$, tel que $\rho(f) \geq 1$.

Exemple 2.19. Supposons que le langage soit composé de la constante o et de la fonction unaire s . On peut choisir, par exemple, les structures suivantes :

1. $D_{\mathcal{M}}$ est l'ensemble des entiers naturels, $o^{\mathcal{M}}$ est le nombre 0 et $s^{\mathcal{M}}$ est la fonction donnant le successeur, c'est-à-dire $s^{\mathcal{M}}(n) = n + 1$;
2. $D_{\mathcal{M}}$ est un ensemble de personnes, $o^{\mathcal{M}}$ c'est Paul et $s^{\mathcal{M}}$ est la fonction donnant le père d'une personne.

Exemple 2.20. Supposons que $\mathcal{S}_f = \{(o, 0), (s, 1)\}$ et $\mathcal{S}_r = \{(\text{Even}, 1)\}$. Comme auparavant, on peut choisir $D_{\mathcal{M}}$ est l'ensemble des entiers naturels, $o^{\mathcal{M}}$ est le nombre 0 et $s^{\mathcal{M}}$ est la fonction donnant le successeur. Pour compléter la définition de \mathcal{S} -structure nous pouvons poser

$$\text{Even}^{\mathcal{M}} := \{n \in \mathbb{N} \mid n \bmod 2 = 1\}.$$

Bien que la \mathcal{S} -structure ainsi définie puisse apparaître bien drôle (ou inappropriée), la définition de cette structure n'est pas incorrecte : *rien nous oblige à donner à un symbole une interprétation par défaut.*

La situation est assez différente quand le symbole d'égalité est parmi les symboles de relation. Dans ce cas, on interprétera, *par défaut*, la relation d'égalité sur l'image de la fonction diagonale :

$$=^{\mathcal{M}} := \{(d, d) \mid d \in D_{\mathcal{M}}\}.$$

Exemple 2.21. Supposons que le langage est composé du symbole de relation B d'arité 2.

On peut choisir par exemple les interprétations suivantes :

1. $D_{\mathcal{M}}$, le domaine, est l'ensemble des sommets du graphe



et B est interprété comme la relation "flèche" : $B^{\mathcal{M}} = \{(a, b), (b, c), (c, d), (d, a)\}$

2. $D_{\mathcal{M}}$, le domaine, est le même qu'auparavant, mais maintenant B est interprété comme la relation "ne pas être voisin direct" : $B^{\mathcal{M}} = \{(b, d), (d, b), (a, c), (c, a)\}$. Cette structure est différente de la précédente, même si elles partagent le même domaine.
3. Le domaine est l'ensemble des triangles et B est la relation "avoir la même aire"

Exemple 2.22. Supposons que $\rho(R) \leq 2$ pour tout $R \in \mathcal{S}_r$, et que $\rho(f) \leq 1$ pour tout $f \in \mathcal{S}_f$. On peut représenter une \mathcal{S} -structure \mathcal{M} comme un (sorte de) graphe orienté étiqueté :

- les noeuds du graphe sont les éléments du domaine $D_{\mathcal{M}}$;
- un noeud $d \in D_{\mathcal{M}}$ est étiqueté par $P \in \mathcal{S}_r$ si $d \in P^{\mathcal{M}}$;

- un noeud $d \in D_{\mathcal{M}}$ est étiqueté par un symbole de constante $c \in \mathcal{S}_f$ si $d = c^{\mathcal{M}}$;
- on met une arête de d vers d' si $(d, d') \in R^{\mathcal{M}}$, pour un quelque symbole de relation $R \in \mathcal{S}_r$; une arête $d \rightarrow d'$ est étiqueté par les symboles $R \in \mathcal{S}_r$ tels quel $(d, d') \in R$;
- on met une arête pointillée de d vers d' si $f(d) = d'$; une arête pointillée $d \rightarrow d'$ est étiqueté par les symboles $f \in \mathcal{S}_f$ tels quel $f(d) = d'$;

Par exemple, si $\mathcal{S}_r = \{(P, 2)\}$ et $\mathcal{S}_f = \{(f, 1)\}$, alors la structure $\mathcal{M} = \langle D_{\mathcal{M}}, P^{\mathcal{M}}, f^{\mathcal{M}} \rangle$ avec

- $D_{\mathcal{M}} = \{a, b, c\}$,
- $P^{\mathcal{M}} = \{(a, b), (b, c), (c, a)\}$,
- $f^{\mathcal{M}} : a \mapsto a, b \mapsto b, c \mapsto a$;

peut se représenter comme le graphe étiqueté de la figure 2.1.

Exemple 2.23. Voici quelques exemples de structures importantes :

Domaine	Fonctions	Relations	Nom
\mathbb{N}	$0, s, +$	$=, \leq$	Arithmétique de Presburger
\mathbb{N}	$0, s, +, \times$	$=, \leq$	Arithmétique de Peano
\mathbb{R}	$0, s, +, \times$	$=, \leq$	Théorie des réels
$\{0\}$	\emptyset	$\{p_0, \dots, p_n, \dots\}$	Structure propositionnelle
$\mathcal{T}_{\mathcal{S}_f}(\emptyset)$	$\hat{\mathcal{S}}_f$	$\hat{\mathcal{S}}_r$	Modèle de Herbrand

Dans le cas des modèle de Herbrand, le domaine est l'ensemble des termes définis sur un ensemble de variables vide. Une fonction $\hat{f} \in \hat{\mathcal{S}}_f$ d'arité n associe aux termes t_1, \dots, t_n le nouveau terme $\hat{f}(t_1, \dots, t_n)$, et les relations sont quelconques.

2.4.2 Evaluation (des termes et) des formules

Comme pour le calcul des propositions, nous allons définir l'interprétation d'une formule en fonction de l'interprétation des formules élémentaires (ici les formules atomiques). Pour qu'une formule soit évaluable à vrai ou faux (1 ou 0), il faut non seulement dire comment s'interprètent les prédicats et les symboles de fonctions, (ce qui est l'analogue d'interpréter les propositions pour le calcul propositionnel) mais aussi ce que valent les variables : en effet les formules peuvent contenir des variables libres, et on a besoin de connaître leur valeur pour que la formule soit évaluable. Bien sûr la valeur des variables liées n'intervient en rien dans le calcul de la valeur d'une formule.

Par exemple pour connaître la valeur de $P(x, y)$ il faut non seulement connaître la signification de P (donnée par la structure), mais aussi la valeur de x et de y qui sera donnée par une **valuation**. Dans la formule $\exists x P(x, y)$ il faut connaître la valeur P et celle de y (mais celle de x n'a aucune importance). Cependant, comme la valeur de $\exists x P(x, y)$ va être définie en fonction de la valeur de $P(x, y)$, on fera aussi intervenir la valeur de x , ou plus précisément les valeurs de $P(x, y)$ pour toutes les valeurs de x .

Nous allons donc définir la valeur d'une formule φ d'un langage (\mathcal{S}, X) en fonction d'une \mathcal{S} -structure \mathcal{M} et d'une valuation des variables.

Définition 2.24. Une **valuation** de l'ensemble X des variables individuelles dans une structure \mathcal{M} est une fonction \mathcal{V} de l'ensemble X vers le domaine de \mathcal{M} , soit $\mathcal{V} : X \rightarrow D_{\mathcal{M}}$.

On définit maintenant la valeur $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$ d'une formule φ en fonction d'une structure \mathcal{M} et d'une valuation \mathcal{V} . On commence naturellement par donner la valeur des termes.

Définition 2.25. Soit \mathcal{M} une \mathcal{S} -structure et \mathcal{V} une valuation de X dans $D_{\mathcal{M}}$; la valeur d'un terme $t \in \mathcal{T}_{\mathcal{S}_f}(X)$, notée $\llbracket t \rrbracket_{\mathcal{M}, \mathcal{V}}$ est un élément de $D_{\mathcal{M}}$ défini (par induction) par :

- pour toute variable $x \in X$, $\llbracket x \rrbracket_{\mathcal{M}, \mathcal{V}} = \mathcal{V}(x)$;
- pour tout $f \in \mathcal{S}_f$ d'arité $n \geq 0$, pour tous termes t_1, \dots, t_n ,

$$\llbracket f(t_1, \dots, t_n) \rrbracket_{\mathcal{M}, \mathcal{V}} := f^{\mathcal{M}}(\llbracket t_1 \rrbracket_{\mathcal{M}, \mathcal{V}}, \dots, \llbracket t_n \rrbracket_{\mathcal{M}, \mathcal{V}}).$$

Remarque 2.26. La définition de l'évaluation pour un terme construit via un symbole de fonction peut se séparer en deux cas, selon l'arité du symbole :

- pour toute constante $c \in \mathcal{S}_f$ (c'est-à-dire symbole de fonction d'arité 0), $\llbracket c() \rrbracket_{\mathcal{M}, \mathcal{V}} = \llbracket c \rrbracket_{\mathcal{M}, \mathcal{V}} := c^{\mathcal{M}}$;
- pour tout $f \in \mathcal{S}_f$ d'arité $n \geq 1$, pour tous termes t_1, \dots, t_n , $\llbracket f(t_1, \dots, t_n) \rrbracket_{\mathcal{M}, \mathcal{V}} := f^{\mathcal{M}}(\llbracket t_1 \rrbracket_{\mathcal{M}, \mathcal{V}}, \dots, \llbracket t_n \rrbracket_{\mathcal{M}, \mathcal{V}})$.

Si \mathcal{M} et \mathcal{V} sont fixées, toute formule atomique prend la valeur 0 ou 1, selon la définition formelle suivante, qui suit l'intuition :

Définition 2.27. La valeur $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$ d'une formule atomique φ est définie par :

- pour tout $R \in \mathcal{S}_r$ d'arité n , pour tous termes t_1, \dots, t_n ,

$$\llbracket R(t_1, \dots, t_n) \rrbracket_{\mathcal{M}, \mathcal{V}} = 1 \quad \text{ssi} \quad (\llbracket t_1 \rrbracket_{\mathcal{M}, \mathcal{V}}, \dots, \llbracket t_n \rrbracket_{\mathcal{M}, \mathcal{V}}) \in R^{\mathcal{M}}.$$

Pour évaluer une formule contenant des quantificateurs, il nous faudra la notion de variante d'une valuation.

Notation 2.28. Nous allons noter $\mathcal{V}[x := a]$ la valuation \mathcal{V}' telle que $\mathcal{V}'(x) = a$ et $\mathcal{V}'(y) = \mathcal{V}(y)$ pour tout $y \in X \setminus \{x\}$. Autrement, pour tout $y \in X$:

$$\mathcal{V}[x := a](y) = \begin{cases} a, & \text{si } y = x, \\ \mathcal{V}(y), & \text{sinon.} \end{cases}$$

On dira alors que $\mathcal{V}[x := a]$ est une **variante** en x de \mathcal{V} .

Enfin, la valeur d'une formule est définie par induction sur la structure de la formule :

Définition 2.29 (Valeur d'une formule). Etant donné un langage \mathcal{S} et une formule φ de $\mathcal{F}_{\text{po}}(\mathcal{S})$, la valeur de φ pour la \mathcal{S} -structure \mathcal{M} et la valuation \mathcal{V} est notée $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$ et est définie de la façon suivante :

- $\llbracket \varphi \wedge \psi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ ssi $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ et $\llbracket \psi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$;
- $\llbracket \varphi \vee \psi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ ssi $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ ou $\llbracket \psi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$;
- $\llbracket \neg \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ ssi $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 0$;
- $\llbracket \varphi \Rightarrow \psi \rrbracket_{\mathcal{M}, \mathcal{V}} = 0$ ssi $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ et $\llbracket \psi \rrbracket_{\mathcal{M}, \mathcal{V}} = 0$;
- $\llbracket \forall x \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ si et seulement si pour tout $a \in D_{\mathcal{M}}$, $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 1$;
- $\llbracket \exists x \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ si et seulement s'il existe $a \in D_{\mathcal{M}}$ tel que $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 1$.

Remarque 2.30. Le quantificateur universel peut se considérer comme une sorte de grande conjonction. En fait, on a

$$\llbracket \forall x. \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = \min\{ \llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} \mid a \in D_{\mathcal{M}} \}.$$

De façon similaire, le quantificateur existentiel est une sorte disjonction :

$$\llbracket \exists x. \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = \max\{ \llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} \mid a \in D_{\mathcal{M}} \}.$$

Par ailleurs, il devient possible remplacer un quantificateur universel par une conjonction (de façon à simuler la logique du premier ordre par la logique propositionnelle) seulement si le domaine $D_{\mathcal{M}}$ est fini, et en plus il est fixé. Que dire si $D_{\mathcal{M}}$ est \mathbb{N} (ici, le domaine est infini) ou si on se pose la question si $\forall x(x = x)$ est vraie dans n'importe quelle structure (ici, on ne peut pas fixer le domaine) ?

On notera souvent $\mathcal{M}, \mathcal{V} \models \varphi$ à la place de $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$.

Remarque 2.31. La valeur de $\llbracket \forall x \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$ ou $\llbracket \exists x \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$ ne dépend pas de $\mathcal{V}(x)$. Par suite, la valeur de $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$, où φ est une formule quelconque ne dépend pas de $\mathcal{V}(x)$ lorsque x n'est pas une variable libre. En particulier, si φ est une formule close (sans variables libres), alors $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$ ne dépend pas de \mathcal{V} et par conséquent on écrira $\llbracket \varphi \rrbracket_{\mathcal{M}}$ à la place de $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$ et $\mathcal{M} \models \varphi$ à la place

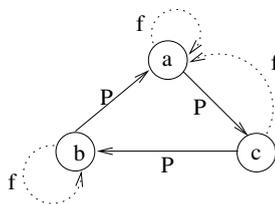


FIGURE 2.1 – Structure en forme de graphe étiqueté

de $\mathcal{M}, \mathcal{V} \models \varphi$.

L'ordre d'apparition des quantificateurs dans une formule est important. Il détermine le sens de la formule.

Par exemple, si nous donnons au prédicat $p(x, y)$ la signification " x aime y ", alors, nous obtenons des significations différentes pour la formule $Q_1 x Q_2 y p(x, y)$ (où Q_1 et Q_2 sont des quantificateurs).

- $\forall x \forall y p(x, y)$ tout le monde aime tout le monde
- $\exists x \forall y p(x, y)$ il existe des personnes qui aiment tout le monde
- $\exists y \forall x p(x, y)$ il existe des personnes aimées de tous
- $\forall x \exists y p(x, y)$ toute personne aime quelqu'un
- $\forall y \exists x p(x, y)$ toute personne est aimée par quelqu'un
- $\exists x \exists y p(x, y)$ il y a une personne qui aime quelqu'un.

On peut remarquer que les seuls cas où on peut échanger l'ordre des quantificateurs sans modifier le sens de la formule sont ceux où les quantificateurs sont identiques : $\exists x \exists y \varphi \equiv \exists y \exists x \varphi$ et $\forall x \forall y \varphi \equiv \forall y \forall x \varphi$. C'est pourquoi en mathématique on écrit souvent $\exists xy \varphi$ ou $\forall xy \varphi$.

Exemple 2.32. Considérons la formule

$$\varphi := \forall x \exists y (P(x, f(y)) \vee P(y, f(x))),$$

à évaluer dans $\mathcal{M} = \langle D_{\mathcal{M}}, P^{\mathcal{M}}, f^{\mathcal{M}} \rangle$ où :

- $D_{\mathcal{M}} = \{a, b, c\}$,
- $P^{\mathcal{M}} = \{(a, b), (b, c), (c, a)\}$,
- $f^{\mathcal{M}} : a \mapsto a, b \mapsto b, c \mapsto a$.

Cette structure, étant sur un langage relationnel d'arité au plus 2 et sur un langage fonctionnel d'arité au plus 1, est représentée en forme de graphe étiqueté en figure 2.1.

Pour calculer $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}}$ on peut d'abord considérer toutes les valuations possibles de x et de y , soit 9 valuations. Pour chaque valuation, nous pouvons évaluer la formule $P(x, f(y)) \vee P(y, f(x))$, en évaluant d'abord les termes $x, f(x), y, f(y)$, pour ensuite évaluer la disjonction selon les règles usuelles de la logique propositionnelle. Nous avons donc :

$\mathcal{V}_{aa} : x = a, y = a :$

$$\llbracket P(x, f(y)) \vee P(y, f(x)) \rrbracket_{\mathcal{M}, \mathcal{V}_{aa}} = \max(\llbracket P(x, f(y)) \rrbracket_{\mathcal{M}, \mathcal{V}_{aa}}, \llbracket P(y, f(x)) \rrbracket_{\mathcal{M}, \mathcal{V}_{aa}}) = 0$$

car $\llbracket f(y) \rrbracket_{\mathcal{M}, \mathcal{V}_{aa}} = \llbracket f(x) \rrbracket_{\mathcal{M}, \mathcal{V}_{aa}} = a$ et donc

$$\llbracket P(x, f(y)) \rrbracket_{\mathcal{M}, \mathcal{V}_{aa}} = \llbracket P(y, f(x)) \rrbracket_{\mathcal{M}, \mathcal{V}_{aa}} = 0, \quad \text{car } (a, a) \notin P^{\mathcal{M}}.$$

Dans la suite, nous allons abréger l'exposition, tous ces calculs seront sous-entendus. Avec un abus de notation, nous allons simplement écrire :

$$\llbracket P(x, f(y)) \vee P(y, f(x)) \rrbracket_{\mathcal{M}, \mathcal{V}_{aa}} = \llbracket P(a, a) \vee P(a, a) \rrbracket_{\mathcal{M}} = 0.$$

$\mathcal{V}_{ab} : x = a, y = b :$

$$\llbracket P(x, f(y)) \vee P(y, f(x)) \rrbracket_{\mathcal{M}, \mathcal{V}_{ab}} = \llbracket P(a, b) \vee P(b, a) \rrbracket_{\mathcal{M}} = 1;$$

$\mathcal{V}_{ac} : x = a, y = c :$

$$\llbracket (P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_{ac}} = \llbracket P(a, a) \vee P(c, a) \rrbracket_{\mathcal{M}} = 1;$$

$\mathcal{V}_{ba} : x = b, y = a :$

$$\llbracket (P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_{ba}} = \llbracket P(b, a) \vee P(a, b) \rrbracket_{\mathcal{M}} = 1;$$

$\mathcal{V}_{bb} : x = b, y = b :$

$$\llbracket (P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_{bb}} = \llbracket P(b, b) \vee P(b, b) \rrbracket_{\mathcal{M}} = 0;$$

$\mathcal{V}_{bc} : x = b, y = c :$

$$\llbracket (P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_{bc}} = \llbracket P(b, a) \vee P(c, b) \rrbracket_{\mathcal{M}} = 0;$$

$\mathcal{V}_{ca} : x = c, y = a :$

$$\llbracket (P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_{ca}} = \llbracket P(c, a) \vee P(a, a) \rrbracket_{\mathcal{M}} = 1;$$

$\mathcal{V}_{cb} : x = c, y = b :$

$$\llbracket (P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_{cb}} = \llbracket P(c, b) \vee P(b, a) \rrbracket_{\mathcal{M}} = 0;$$

$\mathcal{V}_{cc} : x = c, y = c :$

$$\llbracket (P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_{cc}} = \llbracket P(c, a) \vee P(c, a) \rrbracket_{\mathcal{M}} = 1.$$

Commençons par étudier les valeurs possibles de la sous-formule $\exists y(P(x, f(y)) \vee P(y, f(x)))$:

$\mathcal{V}_a : x := a :$

$$\llbracket \exists y(P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_a} = 1, \quad \text{car } \llbracket P(x, f(y)) \vee P(y, f(x)) \rrbracket_{\mathcal{M}, \mathcal{V}_{ab}} = 1.$$

Donc pour toute valuation \mathcal{V} telle que $\mathcal{V}(x) = a$, $\llbracket \exists y(P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$.

$\mathcal{V}_b : x := b :$

$$\llbracket \exists y(P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_b} = 1, \quad \text{car } \llbracket P(x, f(y)) \vee P(y, f(x)) \rrbracket_{\mathcal{M}, \mathcal{V}_{ba}} = 1.$$

Donc pour toute valuation \mathcal{V} telle que $\mathcal{V}(x) = b$, $\llbracket \exists y(P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$.

$\mathcal{V}_c : x := c :$

$$\llbracket \exists y(P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}_c} = 1, \quad \text{car } \llbracket P(x, f(y)) \vee P(y, f(x)) \rrbracket_{\mathcal{M}, \mathcal{V}_{ca}} = 1.$$

Donc pour toute valuation \mathcal{V} telle que $\mathcal{V}(x) = c$, $\llbracket \exists y(P(x, f(y)) \vee P(y, f(x))) \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$.

Donc pour toute valuation \mathcal{V} , nous avons $\mathcal{M}, \mathcal{V} \models \exists y(P(x, f(y)) \vee P(y, f(x)))$, i.e.,

$$\mathcal{M} \models \forall x \exists y (P(x, f(y)) \vee P(y, f(x))).$$

Remarque 2.33. Observons que la notation $P(a, a)$ et les notations similaires sont impropres, mais très utiles en pratique, et aussi très utilisés par les logiciens!!! En fait, $P(a, a)$ n'est pas une formule (atomique) du premier ordre, car ici a n'est pas un terme, mais plutôt un élément du domaine $D_{\mathcal{M}}$. Pour pouvoir justifier cette notation il faut ajouter au langage un symbole de constante c_a , pour tout élément du domaine $a \in D_{\mathcal{M}}$; il faut de plus étendre l'interprétation en posant $c_a^{\mathcal{M}} = a$.

Vocabulaire

Définition 2.34 (Modèle). Soit $\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$ une formule close (i.e., qui ne contient pas de variable libre) et \mathcal{M} une \mathcal{S} -structure. La structure \mathcal{M} est un **modèle** de φ si $\mathcal{M} \models \varphi$.

Soit $\Gamma \subseteq \mathcal{F}_{\text{po}}(\mathcal{S})$ un ensemble de formules closes (i.e., qui ne contient pas de variable libre) et \mathcal{M} une \mathcal{S} -structure. La structure \mathcal{M} est un **modèle** de Γ si $\mathcal{M} \models \varphi$ pour tout $\varphi \in \Gamma$.

Définition 2.35 (Tautologie). Une formule close $\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$ est une **tautologie** si $\mathcal{M} \models \varphi$ pour toute \mathcal{S} -structure \mathcal{M} .

Définition 2.36 (Formule insatisfaisable). Une formule close $\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$ est **insatisfaisable** si elle n'a pas de modèle.

Définition 2.37 (Conséquence logique). Une formule close φ est **conséquence logique** d'un ensemble de formules closes Γ si tout modèle de Γ est un modèle de φ . On écrit alors $\Gamma \models \varphi$.

Définition 2.38 (Théorie). Une **théorie** est l'ensemble des conséquences logiques d'un ensemble de formules closes.

Par exemple, la théorie des groupes est l'ensemble des formules logiques qui sont vraies dans tous les groupes.

Définition 2.39 (Equivalence). Deux formules φ et ψ de $\mathcal{F}_{\text{po}}(\mathcal{S})$ sont **équivalentes** (noté $\varphi \equiv \psi$) si pour toute \mathcal{S} -structure \mathcal{M} et toute valuation $\mathcal{V} : X \rightarrow D_{\mathcal{M}}$, on $\mathcal{M}, \mathcal{V} \models \varphi$ ssi $\mathcal{M}, \mathcal{V} \models \psi$.

Attention : on peut parler d'équivalence entre deux formules même quand celles-ci ne sont pas de formules closes.

On peut par ailleurs noter que φ et ψ sont équivalentes lorsque $\mathcal{M}, \mathcal{V} \models \varphi \Leftrightarrow \psi$ pour tout \mathcal{M} et \mathcal{V} , de façon qu'elles sont équivalentes ssi $\bar{\forall}(\varphi \Leftrightarrow \psi)$ est une tautologie. Ici, $\bar{\forall}(\varphi)$ est la clôture universelle de la formule φ , obtenue de φ en lui ajoutant une suite de quantificateurs universels $\forall x_1 \forall x_2 \dots \forall x_n$, où x_1, \dots, x_n est la liste des variables libres de φ .

2.5 Manipulation de formules

2.5.1 Substitution de variables

Dans la suite, nous allons préciser ce que veut dire qu'une formule $\psi \in \mathcal{F}_{\text{po}}(\mathcal{S})$ est obtenue d'une formule $\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$ en substituant toute occurrence d'une variable libre x par un terme (ce qui sera noté par $\psi = \varphi_{\{x \rightarrow t\}}$).

Remarque 2.40. Bien que la notion soit intuitive, il ne faut pas que des variables libres deviennent liées par cette substitution. Considérons ce qu'il se passe si l'on substitue de façon naïve x par le terme y dans $\exists yR(x, y)$. La formule $\exists yR(y, y)$ n'est pas le résultat souhaité. On souhaite plutôt avoir comme résultat la formule suivante : $\exists zR(y, z)$. On devine donc la nécessité de renommer les variables de sorte qu'une substitution ne crée pas des nouvelles variables liées.

Si σ est la substitution $\{x_1 \rightarrow t_1, \dots, x_n \rightarrow t_n\}$, on note φ_σ la formule φ dans laquelle toutes les occurrences libres de x_1, \dots, x_n ont été remplacées respectivement par t_1, \dots, t_n . Pour définir formellement la notion de substitution dans une formule, on commence par la définir pour les termes. Soit t un terme de $\mathcal{T}_{\mathcal{S}}(X)$ et σ une substitution :

- si $t = x$ et $x \in \text{Dom}(\sigma)$ alors $t_\sigma = \sigma(x)$, (et $t_\sigma = x$ si $x \notin \text{Dom}(\sigma)$) ;
- si $t = c$ alors $t_\sigma = c$;
- si $t = f(t_1, \dots, t_n)$ alors $t_\sigma = f(t_{1\sigma}, \dots, t_{n\sigma})$.

On définit maintenant la substitution par récurrence sur la structure de la formule :

$$R(t_1, \dots, t_n)_\sigma := R(t_{1\sigma}, \dots, t_{n\sigma}),$$

$$(\varphi \circ \psi)_\sigma := (\varphi_\sigma) \circ (\psi_\sigma), \text{ pour tout connecteur binaire } \circ,$$

$$(Qx.\varphi)_\sigma := \begin{cases} Qx.(\varphi_{\sigma[x \rightarrow x]}), & \text{si } x \notin \text{Im}(\sigma) \\ Qy.(\varphi_{\{x \rightarrow y\}})_\sigma, & \text{si } x \in \text{Im}(\sigma), \text{ où } y \notin \text{Var}(\varphi) \cup \text{Dom}(\sigma) \cup \text{Im}(\sigma), \end{cases}$$

où $Q \in \{\exists, \forall\}$, et $\sigma[x \rightarrow x]$ est la substitution σ' telle que $\sigma'(x) = x$, et $\sigma'(y) = \sigma(y)$ pour $y \neq x$.

Proposition 2.41. Si $y \notin \text{Var}(\varphi)$, alors $\exists x\varphi \equiv \exists y(\varphi_{\{x \rightarrow y\}})$.

Grâce à cette proposition, on pourra supposer désormais, sans perte de généralité, que les différentes occurrences liées d'une variables sont toutes liées au même quantificateur et qu'aucune variable n'admet à la fois des occurrences libres et des occurrences liées.

2.5.2 Equivalences classiques

Les équivalences données dans le cadre du calcul propositionnel restent vraies. Nous en donnons d'autres ici, les preuves sont laissées en exercice.

- Lois de **conversion** des quantificateurs :

$$\neg \forall x\varphi \equiv \exists x\neg\varphi, \quad \neg \exists x\varphi \equiv \forall x\neg\varphi.$$
- Lois de **distribution** des quantificateurs :

$$\forall x(\varphi \wedge \psi) \equiv (\forall x\varphi \wedge \forall x\psi), \quad \exists x(\varphi \vee \psi) \equiv (\exists x\varphi \vee \exists x\psi).$$
- Lois de **permutation** des quantificateurs de même sorte :

$$\forall x\forall y\varphi \equiv \forall y\forall x\varphi, \quad \exists x\exists y\varphi \equiv \exists y\exists x\varphi.$$

— Lois de **passage** : si x ne figure pas à titre d'occurrence libre dans ψ , on a les lois suivantes :

$$\begin{array}{ll} \forall x(\varphi \wedge \psi) \equiv (\forall x\varphi) \wedge \psi & \exists x(\varphi \wedge \psi) \equiv (\exists x\varphi) \wedge \psi \\ \forall x(\varphi \vee \psi) \equiv (\forall x\varphi) \vee \psi & \exists x(\varphi \vee \psi) \equiv (\exists x\varphi) \vee \psi \\ \forall x(\varphi \Rightarrow \psi) \equiv (\exists x\varphi) \Rightarrow \psi & \exists x(\varphi \Rightarrow \psi) \equiv (\forall x\varphi) \Rightarrow \psi \\ \forall x(\psi \Rightarrow \varphi) \equiv \psi \Rightarrow (\forall x\varphi) & \exists x(\psi \Rightarrow \varphi) \equiv \psi \Rightarrow (\exists x\varphi) \end{array}$$

Remarquez le changement de quantificateur dans les équivalences $\forall x(\varphi \Rightarrow \psi) \equiv (\exists x\varphi) \Rightarrow \psi$ et $\exists x(\varphi \Rightarrow \psi) \equiv (\forall x\varphi) \Rightarrow \psi$.

La règle qui établit l'équivalence entre $\exists x(\varphi \wedge \psi)$ et $(\exists x\varphi) \wedge \psi$ (lorsque x n'est pas une variable libre de ψ) est aussi connue sous le nom de **loi de Frobenius**.

— Lois de **réalphabétisation** (renommage) des variables.

On peut toujours renommer une variable liée et la variable du quantificateur au sein d'une formule. Cependant, le nouveau nom ne doit pas être un nom déjà utilisé pour une variable libre ou liée de la formule.

Par exemple, dans $\exists x(\forall y F(x, y) \Rightarrow (G(x) \vee q))$, on peut opérer deux renommages : on peut renommer d'abord l'occurrence de x dans $F(x, y)$ à t , en obtenant ainsi $\exists x(\forall t F(t, y) \Rightarrow (G(x) \vee q))$, et ensuite renommer le x restant à z ; on obtient $\exists z(\forall t F(t, y) \Rightarrow (G(z) \vee q))$. Soit $\{x \rightarrow t\}$ la substitution qui remplace la variable x par la variable t . La loi de réalphabétisation peut se déduire de la règle élémentaire de réalphabétisation suivante :

$$Qx\varphi \equiv Qt(\varphi_{\{x \rightarrow t\}}) \quad \text{où } Q \in \{\forall, \exists\}$$

pourvu que t n'est pas une variable libre de φ (en particulier, quand t n'a aucune occurrence dans t) et toutes les occurrences de x sont libres dans φ .

Exemple 2.42. Nous pouvons démontrer l'équivalence entre les formules $\neg\forall x\varphi(x)$ et $\exists x\neg\varphi(x)$ de la façon suivante. Soient \mathcal{M} une \mathcal{S} -structure et \mathcal{V} une valuation fixés.

- (a) Supposons que $\llbracket \exists x\neg\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ et montrons que $\llbracket \neg\forall x\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$. De $\max_{a \in D_{\mathcal{M}}} \llbracket \neg\varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 1$, nous déduisons que $\llbracket \neg\varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 1$ pour un quelquel $a \in D_{\mathcal{M}}$, donc $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 0$ et $\llbracket \forall x\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = \min_{a \in D_{\mathcal{M}}} \llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 0$, donc $\llbracket \neg\forall x\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$.
- (b) Supposons, par contre, que $\llbracket \neg\forall x\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$ et montrons que $\llbracket \exists x\neg\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$. On a bien que $\llbracket \forall x\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 0$; depuis

$$0 = \llbracket \forall x\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = \min_{a \in D_{\mathcal{M}}} \llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 0$$

nous déduisons que ce minimum est réalisé : donc $\llbracket \varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 0$ pour un quelquel $a \in D_{\mathcal{M}}$, d'où $\llbracket \neg\varphi \rrbracket_{\mathcal{M}, \mathcal{V}[x:=a]} = 1$, et $\llbracket \exists x\neg\varphi \rrbracket_{\mathcal{M}, \mathcal{V}} = 1$.

Les argumentaires ci-dessus sont acceptés dans un cadre classique. L'argumentaire (b) se révèle par contre insatisfaisante, si depuis une preuve d'existence d'un objet avec une certaine propriété, nous souhaitons être capable de construire l'objet. Supposons, par exemple, que nous avons réussi à montrer que « non pour tout n , si n est premier, alors $n = 2^k - 1$ pour un nombre k ». L'argumentaire (b) prétend qu'il est possible déduire de l'existence d'un nombre n qui n'est pas de la forme $2^k - 1$, mais ne spécifie d'aucune façon comment le construire. Pour cette raison, en *logique intuitionniste* (du premier ordre), qui se construit autour de l'idée qu'une preuve logique d'existence doit donner aussi un moyen de construire un objet témoignant de son existence, la formule $\exists x\neg\varphi \Rightarrow \neg\forall x\varphi$ est considérée comme une tautologie, mais la formule $\neg\forall x\varphi \Rightarrow \exists x\neg\varphi$ ne l'est pas. Voir par exemple [Miq05].

Exercice 2.43. Montrez que les formules $\forall x(\varphi \vee \psi)$ et $\forall x\varphi \vee \forall x\psi$ ne sont pas, en général, équivalentes. Argumentez de façon similaire pour $\exists x(\varphi \wedge \psi)$ et $\exists x\varphi \wedge \exists x\psi$.

2.5.3 Formes Normales

Établir la consistance d'une formule du calcul des prédicats est un problème difficile. On peut alors essayer de travailler sur une version de forme plus simple mais de consistance équivalente à la formule initiale. Nous introduisons dans cette section la forme clausale pour la logique des prédicats. Toute formule de la logique des prédicats du premier ordre admet une représentation sous forme de clause qui préserve sa satisfiabilité. À la différence des formes normales, une clause n'est pas logiquement équivalente à la formule dont elle dérive. Nous décrivons, à présent, les différentes étapes qui mènent à une représentation sous forme clausale.

2.5.3.1 Forme prénex

Définition 2.44 (Forme prénex). Une formule φ est sous **forme prénex** lorsqu'elle a la forme :

$$Q_1x_1Q_2x_2 \dots Q_nx_n\psi$$

où chaque Q_i est un quantificateur (existentiel ou universel) et ψ est sans quantificateurs. La partie $Q_1x_1Q_2x_2 \dots Q_nx_n$ est appelée le **préfixe** et ψ étant qualifiée de **matrice**.

Mettre une formule sous forme prénex consiste donc à renvoyer tous les quantificateurs au début de la formule.

Théorème 2.45 (Forme prénex équivalente). *Pour toute formule $\varphi \in \mathcal{F}_{po}$ il existe une formule équivalente $\psi \in \mathcal{F}_{po}$ en forme prénex.*

L'algorithme de mise sous forme prénex suit les étapes suivantes :

Étape 1 : Renommer les variables de façon à ce qu'aucune variable n'ait d'occurrence libre et liée et d'occurrences liées à des quantificateurs différents ;

Étape 2 : Appliquer tant que possible les substitutions suivantes : substitution du membre droit par le membre gauche pour toutes les lois de passage et de conversion des quantificateurs.

Exemple 2.46. Considérez la formule suivante :

$$\neg \exists x \forall y R(x, y) \wedge \forall x (\exists y R(x, y) \Rightarrow R(x, x))$$

La première étape, renommage des variables, donne la formule suivante :

$$\neg \exists z_0 \forall z_1 R(z_0, z_1) \wedge \forall z_2 (\exists z_3 R(z_2, z_3) \Rightarrow R(z_2, z_3)).$$

Nous appliquons ensuite la deuxième étape :

$$\begin{aligned} & \neg \exists z_0 \forall z_1 R(z_0, z_1) \wedge \forall z_2 (\exists z_3 R(z_2, z_3) \Rightarrow R(z_2, z_2)) \\ & \rightsquigarrow \forall z_0 \neg \forall z_1 R(z_0, z_1) \wedge \forall z_2 (\exists z_3 R(z_2, z_3) \Rightarrow R(z_2, z_2)) \\ & \rightsquigarrow \underbrace{\forall z_0 \exists z_1 \neg R(z_0, z_1)}_{\varphi} \wedge \underbrace{\forall z_2 (\exists z_3 R(z_2, z_3) \Rightarrow R(z_2, z_2))}_{\psi} \\ & \rightsquigarrow \forall z_0 (\underbrace{\exists z_1 \neg R(z_0, z_1)}_{\varphi} \wedge \underbrace{\forall z_2 (\exists z_3 R(z_2, z_3) \Rightarrow R(z_2, z_2))}_{\psi}) \\ & \rightsquigarrow \forall z_0 \exists z_1 (\underbrace{\neg R(z_0, z_1)}_{\psi} \wedge \underbrace{\forall z_2 (\exists z_3 R(z_2, z_3) \Rightarrow R(z_2, z_2))}_{\varphi}) \\ & \rightsquigarrow \forall z_0 \exists z_1 \forall z_2 (\underbrace{\neg R(z_0, z_1)}_{\psi} \wedge (\underbrace{\exists z_3 R(z_2, z_3)}_{\varphi} \Rightarrow \underbrace{R(z_2, z_2)}_{\psi})) \\ & \rightsquigarrow \forall z_0 \exists z_1 \forall z_2 (\underbrace{\neg R(z_0, z_1)}_{\psi} \wedge \underbrace{\forall z_3 (R(z_2, z_3) \Rightarrow R(z_2, z_2))}_{\varphi}) \\ & \rightsquigarrow \forall z_0 \exists z_1 \forall z_2 \forall z_3 (\underbrace{\neg R(z_0, z_1)}_{\psi} \wedge (\underbrace{R(z_2, z_3)}_{\varphi} \Rightarrow R(z_2, z_2))). \end{aligned}$$

La nécessité de renommer les variables peut se comprendre si on essaie d'appliquer les règles de passage sans un renommage préalable. Ainsi :

$$\begin{aligned} & \neg \exists x \forall y R(x, y) \wedge \forall x (\exists y R(x, y) \Rightarrow R(x, x)) \\ & \rightsquigarrow \forall x \exists y \neg R(x, y) \wedge \forall x (\exists y R(x, y) \Rightarrow R(x, x)) \\ & \rightsquigarrow \forall x \exists y \underbrace{(\neg R(x, y))}_{\psi} \wedge \forall x \underbrace{(\exists y R(x, y) \Rightarrow R(x, x))}_{\varphi} \end{aligned}$$

Ici on ne peut pas appliquer la loi de passage, donc on renomme :

$$\rightsquigarrow \forall x \exists y \underbrace{(\neg R(x, y))}_{\psi} \wedge \forall t \underbrace{(\exists y R(t, y) \Rightarrow R(t, t))}_{\varphi}$$

On peut ainsi appliquer la loi de passage :

$$\rightsquigarrow \forall x \exists y \forall t (\neg R(x, y) \wedge (\exists y R(t, y) \Rightarrow R(t, t))) \dots$$

2.5.3.2 Forme de Skolem

Définition 2.47. Une formule est sous **forme de Skolem** lorsqu'elle est sous forme prénexe et qu'elle ne contient que des quantifications universelles.

Exemple 2.48. La formule $\forall x R(x, f(x))$ est sous forme de Skolem. La formule $\forall x \exists y R(x, y)$ est en forme prénexe, mais elle n'est pas sous forme de Skolem, car on trouve un quantificateur existentiel dans le préfixe.

Pour effectuer une *skolémisation*, on part donc d'une formule sous forme prénexe et on « supprime » les quantificateurs existentiels, en appliquant de façon itérée la règle suivante.

Règle de Skolémisation. Elle est de la forme

$$\frac{\exists y \psi}{\psi_{\{y \rightarrow f(z_1, \dots, z_m)\}}}$$

où

- z_1, \dots, z_m sont les variables libres de la formule $\exists y \psi$, et
- f un *nouveau* symbole de fonction (dit de Skolem) d'arité m .

Rappel (cf. Définition 2.14). Une formule est *close* si elle ne contient pas de variables libres.

Exemple 2.49. Considérons une formule close $\varphi = \exists x \psi$ écrite sur un langage \mathcal{S} . La règle de skolémisation donne la formule $\psi_{\{x \rightarrow c\}}$, où c est un nouveau symbole de constante.

Une \mathcal{S} -structure \mathcal{M} est un modèle de φ ssi il existe $a \in D_{\mathcal{M}}$ tel que $\llbracket \psi \rrbracket_{\mathcal{M}, \mathcal{V}}$ où \mathcal{V} est telle que $\mathcal{V}(x) = a$. Considérons maintenant le langage $\mathcal{S}' := \mathcal{S} \cup \{(c, 0)\}$ (où c est le nouveau symbole de constante) et la \mathcal{S}' -structure \mathcal{M}' obtenue depuis \mathcal{M} en interprétant la constante c par $c^{\mathcal{M}'} := a$. Clairement, \mathcal{M}' est un modèle de la formule $\psi_{\{x \rightarrow c\}}$.

Nous avons donc argumenté que si φ a un modèle, alors $\psi_{\{x \rightarrow c\}}$ a un modèle ; l'implication inverse est d'ailleurs aussi vraie. Donc φ est satisfaisable ssi $\psi_{\{x \rightarrow c\}}$ est satisfaisable.

Exemple 2.50. Considérons maintenant la formule $\varphi = \forall y \exists x \psi$ écrite sur un langage \mathcal{S} . Une \mathcal{S} -structure \mathcal{M} est un modèle de φ ssi pour chaque $b \in D_{\mathcal{M}}$ il existe $a \in D_{\mathcal{M}}$ tel que $\llbracket \psi \rrbracket_{\mathcal{M}, [y:=b, x:=a]}$ ssi il existe une fonction $f : D_{\mathcal{M}} \rightarrow D_{\mathcal{M}}$ telle que, pour tout $b \in D_{\mathcal{M}}$, $\llbracket \psi \rrbracket_{\mathcal{M}, [y:=b, x:=f(b)]}$.

Donc φ admet un modèle ssi $\forall y \psi_{\{x \rightarrow f(y)\}}$ a un modèle. Remarquez que la formule $\forall y \psi_{\{x \rightarrow f(y)\}}$ est construite à partir du langage $\mathcal{S} \cup \{(f, 1)\}$.

C'est ce principe qui va être généralisé pour mettre une formule sous forme de Skolem.

Définition 2.51. Une formule universelle (c'est-à-dire une formule close contenant seulement des quantificateurs universels), obtenue par

- mise en forme prénexé, et ensuite
- application itérée de la règle de skolemisation

est appelée **forme de skolem** ou **skolemisée** de φ .

Exemple 2.52.

1. Soit $\varphi = \forall x \exists y P(x, y) \Rightarrow \forall x \exists y P(y, x)$.

La mise sous forme prénexé donne la formule équivalente $\varphi' = \exists x \forall y \forall x' \exists y' (P(x, y) \Rightarrow P(y', x'))$.

La skolemisation donne :

$$\frac{\frac{\exists x \forall y \forall x' \exists y' (P(x, y) \Rightarrow P(y', x'))}{\forall y \forall x' \exists y' (P(c, y) \Rightarrow P(y', x'))}}{\forall x \forall x' (P(c, y) \Rightarrow P(f(y, x'), x'))}$$

2. Considérons la formule $\exists x_1 \forall x_2 \forall x_3 \exists x_4 \forall x_5 \exists x_6 P(x_1, x_2, x_3, x_4, x_5, x_6)$. On obtient sa formule de Skolem de la façon suivante :

$$\frac{\frac{\frac{\exists x_1 \forall x_2 \forall x_3 \exists x_4 \forall x_5 \exists x_6 P(x_1, x_2, x_3, x_4, x_5, x_6)}{\forall x_2 \forall x_3 \exists x_4 \forall x_5 \exists x_6 P(f_1, x_2, x_3, x_4, x_5, x_6)}}{\forall x_2 \forall x_3 \forall x_5 \exists x_6 P(f_1, x_2, x_3, f_4(x_2, x_3), x_5, x_6)}}{\forall x_2 \forall x_3 \forall x_5 P(f_1, x_2, x_3, f_4(x_2, x_3), x_5, f_6(x_2, x_3, x_5))}$$

Dans la formule précédente, $\exists x_1$ n'est précédé par aucun quantificateur universel. C'est pourquoi on introduit une nouvelle constante f_1 .

Remarque 2.53. La version skolemisée d'une formule *ne lui est pas*, en général, *équivalente*. Le langage étant étendu, donc différent, les modèles des deux formules sont des structures pour des langages différents. Néanmoins :

- tout modèle de la formule skolemisée est modèle de la formule initiale ;
- tout modèle de la formule initiale peut s'étendre en un modèle de la formule skolemisée, obtenu en conservant les interprétations des symboles de la signature initiale, et en interprétant correctement les nouveaux symboles de fonction introduits pas la skolemisation ;
- une formule close et sa forme de Skolem sont dites *équisatisfaisables* : si l'une possède un modèle, l'autre également et réciproquement.

Proposition 2.54. Si φ_s est obtenue par skolemisation à partir de φ alors φ_s est satisfaisable si et seulement si φ est satisfaisable.

2.5.3.3 Forme clausale

Définition 2.55 (Forme clausale). Une formule close est sous **forme clausale** si elle est

1. en forme prénexé,
2. elle est universelle (tous ses quantificateurs sont universels), et
3. sa matrice est sous forme normale conjonctive.

En utilisant la mise en forme prénexé, puis la Skolemisation, puis la mise en forme clausale du calcul propositionnel, toute formule peut se transformer dans une formule en forme clausale equisatisfaisable.

Définition 2.56.

- Un **littéral** est une formule atomique ou la négation d'une formule atomique.

— Soit ψ une formule avec $FV(\psi) = \{x_1, \dots, x_n\}$. Sa **fermeture universelle** est la formule

$$\forall x_1 \dots \forall x_n \psi.$$

— Une **clause universelle** ou **clause de premier ordre** est la fermeture universelle d'une disjonction de littéraux.

Lorsqu'une formule est sous forme clausale, on peut ensuite la décomposer en une conjonction de clauses (du premier ordre) en appliquant la règle $\forall x(\varphi \wedge \psi) \equiv (\forall x\varphi) \wedge (\forall x\psi)$. On obtient ainsi un ensemble de clauses (de premier ordre) équisatisfiable à la formule donnée.

Théorème 2.57 (Satisfiabilité d'un ensemble de clauses). *Soit S un ensemble de clauses résultant de la mise sous forme clausale d'une formule φ . Alors φ est satisfiable si et seulement si S est satisfiable.*

Ce théorème forme la base de nombreux démonstrateurs automatiques utilisant une représentation des formules sous forme de clauses. Il établit que la recherche de l'insatisfiabilité d'une formule φ est équivalente à la recherche d'insatisfiabilité de sa représentation sous forme clausale S . Cependant φ et S ne sont pas logiquement équivalentes : seule la satisfiabilité est préservée.

Exemple 2.58. Soit

$$\varphi_0 := \forall x \exists y P(x, y) \Rightarrow \forall x \exists y P(y, x \wedge Q(x)).$$

La mise sous forme prénexée donne la formule

$$\varphi_1 := \exists x \forall y \forall z \exists w ((P(x, y) \Rightarrow P(w, z)) \wedge Q(x)).$$

Par skolémisation, nous obtenons

$$\varphi_2 := \forall y \forall z ((P(c, y) \Rightarrow P(f(y, z), z)) \wedge Q(c)).$$

Nous pouvons ensuite mettre la matrice sous forme normale conjonctive :

$$\varphi_3 := \forall y \forall z ((\neg P(c, y) \vee P(f(y, z), z)) \wedge Q(c)),$$

et obtenir ainsi l'ensemble de clauses :

$$S := \{ \forall y \forall z (\neg P(c, y) \vee P(f(y, z), z)), \forall y \forall z Q(c) \}.$$

Cet ensemble est equisatisfiable avec la formule φ_0 .

2.6 Unification

2.6.1 Substitutions et MGUs

Dans la suite, on fixe \mathcal{S}_f une signature composée de symboles de fonctions et X un ensemble de variables. Nous nous intéresserons pas, dans cette section, aux symboles de relation.

Définition 2.59. Une **substitution** est une fonction $\sigma : X \rightarrow \mathcal{T}_{\mathcal{S}_f}(X)$ telle que l'ensemble $\{x \in X \mid \sigma(x) \neq x\}$ est fini.

Exemple 2.60. Supposons $\mathcal{S}_f = \{(f, 2), (g, 1)\}$ et $X = \{x_0, \dots, x_n, \dots\}$. La fonction σ telle que $\sigma(x_0) = f(g(x_0), x_1)$, $\sigma(x_1) = x_2$, et $\sigma(x_i) = x_i$ pour $i \geq 2$, est une substitution.

On représente (souvent, dans des implémentations sur ordinateur) et on écrit les substitutions par des listes de couples clef,valeur (en jargon informatique, c'est des *listes associatives*), notées d'habitude par :

$$\{x_1 \rightarrow t_1, \dots, x_n \rightarrow t_n\}, \quad \text{ou} \quad [t_1/x_1, \dots, t_n/x_n].$$

La convention est la suivante : étant donnée une telle liste on construit la substitution σ en posant $\sigma(x_i) = t_i$; si une variable x n'apparaît pas dans la liste, alors elle est fixée par σ , c'est-à-dire $\sigma(x) = x$. Ainsi, la substitution de l'exemple 2.60, sera notée par

$$[f(g(x_0), x_1)/x_0, x_2/x_1].$$

Exercice 2.61. Argumentez que toute substitution peut être représentée par une liste associative. Montrez que cette représentation n'est pas unique, c'est-à-dire qu'ils existent plusieurs (même une infinité) de listes représentant la même substitution.

Définition 2.62. Pour tout terme t , on définit l'action de σ sur t comme suit :

$$\begin{aligned} x\sigma &= \sigma(x), \\ f(t_1, \dots, t_n)\sigma &= f(t_1\sigma, \dots, t_n\sigma). \end{aligned}$$

Exemple 2.63. On a

$$f(g(x), y)[z/x, g(y)/y] = f(g(z), g(y)), \quad g(f(x, f(y, x)))[g(w)/x] = g(f(g(w), f(y, g(w)))).$$

Définition 2.64. La substitution **identité** (ou vide) est celle qui fixe toutes les variables. Elle est donc notée par \square . La **composition** de deux substitutions σ et τ , notée $\tau \circ \sigma$, est la substitution définie par :

$$(\tau \circ \sigma)(x) = (x\sigma)\tau. \quad (2.2)$$

Observez que, dans la définition ci-dessus, l'expression à la droite est $(\sigma(x))\tau$.

Exemple 2.65. Soient

$$\sigma = [f(x, y)/x], \quad \tau = [g(y)/x, f(x, z)/y].$$

Calculons $\tau \circ \sigma$:

σ	τ
$x \mapsto f(x, y)$	$\mapsto f(g(y), f(x, z))$
$y \mapsto y$	$\mapsto f(x, z)$
$z \mapsto z$	$\mapsto z$
	\vdots

On a donc

$$\tau \circ \sigma = [f(g(y), f(x, y))/x, f(x, z)/y].$$

Exercice 2.66. En généralisant l'exemple 2.65, proposez un algorithme qui calcule la composition de deux substitutions σ et τ passées en paramètre. Les substitutions, en entrée et en sortie, seront représentées par des listes associatives.

Exercice 2.67. Prouvez les relations suivantes :

$$\begin{aligned} t \square &= t, \\ t(\tau \circ \sigma) &= (t\sigma)\tau. \end{aligned} \quad (2.3)$$

Prouvez ensuite que la composition de substitutions est associative, et que la substitution identité est son un élément neutre.

Remarque 2.68. La composition $\tau \circ \sigma$ de deux substitution σ, τ revient à une sorte de composition fonctionnelle—car on applique d'abord σ et puis τ . Cela justifie de dénoter cette composition par le symbole usuel (le symbole \circ) de la composition de fonctions.

Par ailleurs, il est costume en logique (et il s'avère éclaircissant) d'écrire la substitution à la droite du terme dans l'application d'un substitution à un terme, D'ici les relations (2.2) et (2.3) qui, en renversant gauche et droite, pourraient apparaître à première vue un peu bizarres.

Définition 2.69. Soient σ et τ deux substitutions. On dit que σ est **plus générale** que τ (et on écrit $\sigma \leq \tau$), s'il existe une substitution ρ telle que $\tau = \rho \circ \sigma$.

Exemple 2.70. Soit

$$\sigma = [f(w, x)/x, z/y], \quad \tau = [f(g(y), x)/x, c/y, c/z, g(y)/w].$$

On a alors $\sigma \leq \tau$, à cause de

$$\rho = [c/z, g(y)/w].$$

En fait, le calcul de la composition donne :

	σ		ρ	
x	\mapsto	$f(w, x)$	\mapsto	$f(g(y), x)$
y	\mapsto	z	\mapsto	c
z	\mapsto	z	\mapsto	c
w	\mapsto	w	\mapsto	$g(y)$

Définition 2.71. Un **problème d'unification** est une liste $(s_1, t_1), \dots, (s_n, t_n)$ avec $s_i, t_i \in \mathcal{T}_{S_i}(X)$. Une solution de ce problème—appelé **unificateur** de $(s_1, t_1), \dots, (s_n, t_n)$ —est une substitution σ telle que $s_i\sigma = t_i\sigma$, pour $i = 1, \dots, n$. On notera $\text{Unif}[(s_1, t_1), \dots, (s_n, t_n)]$ l'ensemble des unificateurs $(s_1, t_1), \dots, (s_n, t_n)$.

Exemple 2.72.

1. La substitution

$$\sigma = [g(z)/x, g(z)/y].$$

est un unificateur de $(f(x, g(z)), f(g(z), y))$, car

$$f(x, g(z))[g(z)/x, g(z)/y] = f(g(z), g(z)) = f(g(z), y)[g(z)/x, g(z)/y].$$

2. Nous avons $\text{Unif}[(f(x, y), g(z))] = \emptyset$. De même, $\text{Unif}[(x, g(x))] = \emptyset$.

Le but de cette section est de montrer le résultat suivant :

Proposition 2.73. Si $\text{Unif}[(s_1, t_1), \dots, (s_n, t_n)] \neq \emptyset$, alors il existe $\sigma \in \text{Unif}[(s_1, t_1), \dots, (s_n, t_n)]$ tel que $\sigma \leq \tau$ pour tout $\tau \in \text{Unif}[(s_1, t_1), \dots, (s_n, t_n)]$.

On appelle un tel σ un **unificateur plus général** des couples $(s_1, t_1), \dots, (s_n, t_n)$. On dira aussi que σ un MGU des couples $(s_1, t_1), \dots, (s_n, t_n)$, où MGU est un acronyme de l'anglais « Most General Unifier ».

Exemple 2.74. $\tau = [g(f(w))/x, g(f(w))/y] \in \text{Unif}[(f(x, g(z)), f(g(z), y))]$, mais τ n'est pas un MGU de ce problème. En fait, $\sigma = [g(z)/x, g(z)/y]$ est un MGU, et on a $\sigma \leq \tau$, car $\tau = \rho \circ \sigma$, avec $\rho = [f(w)/z]$.

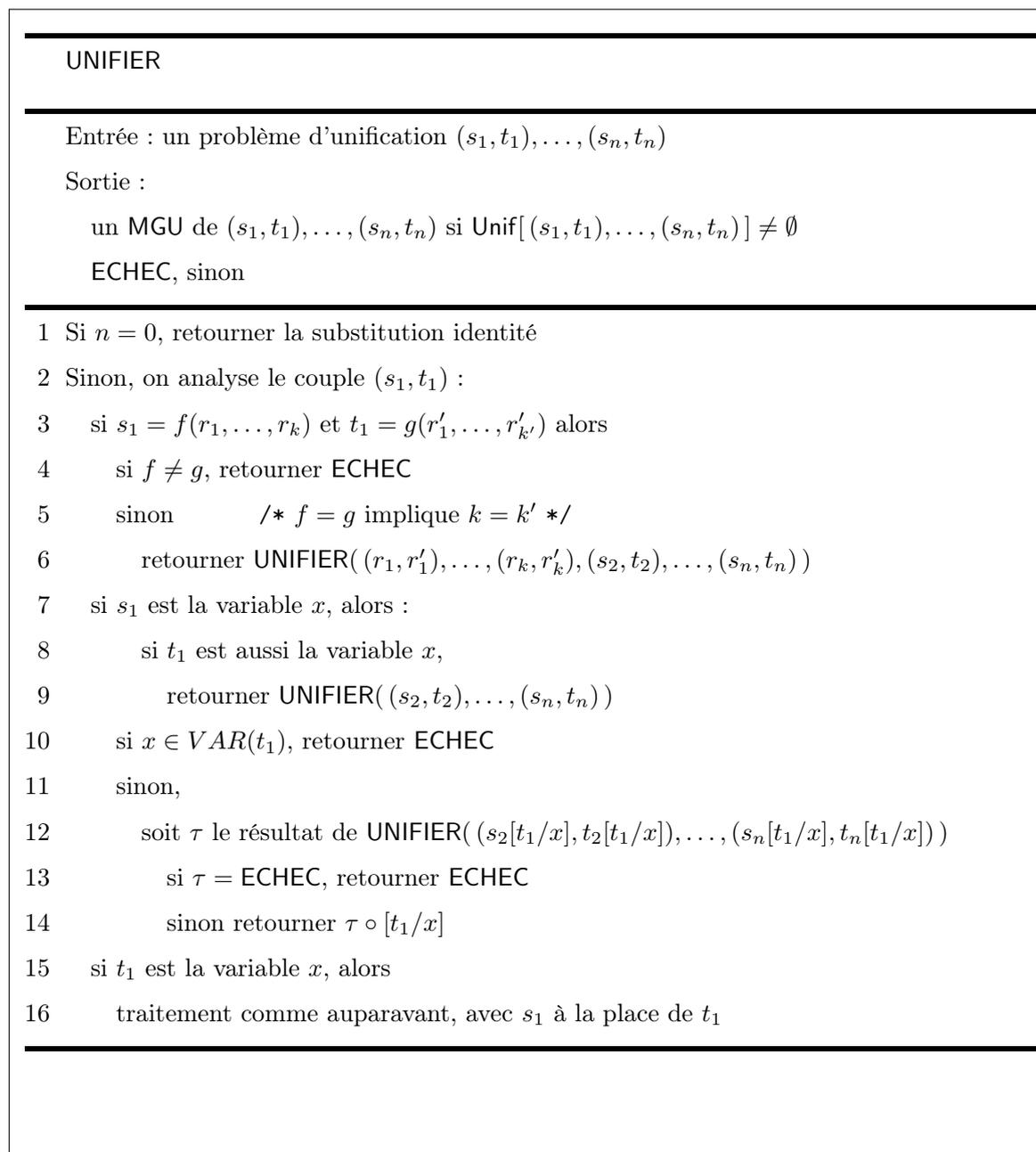


FIGURE 2.2 – Algorithme d'unification

2.6.2 Algorithme d'unification

L'algorithme d'unification est illustré en figure 2.6.2. Nous donnons dans la suite des exemples de calcul de cet algorithme sur des problèmes d'unification.

Exemple 2.75. Considérez le problème suivant :

$$(f(x, g(z)), f(g(z), x)), (x, g(z)).$$

L'algorithme marche de la façon suivante :

Ligne appel récursif (ou return)	Entrée	Pile des résultats partiels
	$(f(x, g(z)), f(g(z), x)), (x, g(z))$	
6	$(x, g(z)), (g(z), x), (x, g(z))$	
12	$(g(z), g(z)), (g(z), g(z))$	$[g(z)/x]$
6	$(z, z), (g(z), g(z))$	$[g(z)/x]$
9	$(g(z), g(z))$	$[g(z)/x]$
6	(z, z)	$[g(z)/x]$
9		$[g(z)/x]$
1		$[] \circ [g(z)/x]$

Exercice 2.76. Exercez vous maintenant avec les problèmes suivants :

1. $(f(g(k(x)), y), f(y, g(x))),$
2. $(f(g(x), x), f(y, g(z))), (g(x), y)),$
3. $(f(y, k(y), g(x)), f(k(x), k(y), y)).$

Terminaison. Définissons la complexité d'un terme comme suit :

$$\begin{aligned} \#(x) &= 1, \\ \#(f(t_1, \dots, t_n)) &= 1 + \sum_{i=1, \dots, n} \#t_i. \end{aligned}$$

La complexité d'un problème (que nous noterons par le même symbole $\#$) est un couple de nombres entiers (non-négatifs) qui se définit comme suit :

$$\#((s_1, t_1), \dots, (s_n, t_n)) = (\text{card}(\bigcup_{i=1, \dots, n} \text{Var}(s_i) \cup \text{Var}(t_i)), \sum_{i=1, \dots, n} \#(s_i) + \#(t_i)).$$

Le lecteur notera qu'à chaque appel récursif, le problème en paramètre a complexité strictement plus petite par rapport à l'ordre lexicographique sur $\mathbb{N} \times \mathbb{N}$. Donc, l'algorithme ne peut pas faire une suite infinie d'appels récursifs, et il termine.

2.6.3 Correction et complétude

Nous souhaitons prouver les propositions suivantes.

Proposition 2.77 (Correction). *Soit π un problème d'unification. Si $\text{UNIFIER}(\pi)$ retourne ECHEC, alors $\text{Unif}[\pi] = \emptyset$; si $\text{UNIFIER}(\pi)$ retourne une substitution σ , alors $\sigma \in \text{Unif}[\pi]$ et, de plus, σ est un MGU de π .*

Proposition 2.78 (Complétude). *Soit π un problème d'unification. Si $\text{Unif}[\pi] = \emptyset$, alors $\text{UNIFIER}(\pi)$ retourne ECHEC; si $\text{Unif}[\pi] \neq \emptyset$, alors $\text{UNIFIER}(\pi)$ retourne un MGU de π .*

Afin de prouver ces deux propositions, introduisons quelques notations, ainsi que la Proposition 2.79, qui est le résultat nécessaire le moins évident à démontrer.

$$- \Delta = \{(t, t) \mid t \in \mathcal{T}_{S_r}(X)\} \text{ et } \Delta^n = \underbrace{\Delta \times \dots \times \Delta}_{n\text{-fois}}$$

— si $\pi = (s_1, t_1), \dots, (s_n, t_n)$, alors $\ell(\pi) = n$ et $\pi_\sigma = (s_1\sigma, t_1\sigma), \dots, (s_n\sigma, t_n\sigma)$.

Avec cette notation remarquez que

$$\sigma \in \text{Unif}[\pi] \text{ ssi } \pi_\sigma \in \Delta^{\ell(\pi)}.$$

Proposition 2.79. *Soient π et ψ deux problèmes d'unification, soit σ un MGU de π . Alors*

$$\text{Unif}[\pi, \psi] = \{ \rho \circ \sigma \mid \rho \in \text{Unif}[\psi_\sigma] \}. \quad (2.4)$$

Par conséquent, si ρ est un MGU de ψ_σ , alors $\tau = \rho \circ \sigma$ un MGU de π, ψ .

Démonstration. Observons que si $\tau \in \text{Unif}[\pi, \psi]$ alors $\tau \in \text{Unif}[\pi]$ et, car σ est un MGU de π , $\sigma \leq \tau$, c'est-à-dire $\tau = \rho \circ \sigma$ pour une substitution ρ . Car $\tau \in \text{Unif}[\psi]$, on remarquera que

$$(\psi_\sigma)_\rho = \psi_{\rho \circ \sigma} = \psi_\tau \in \Delta^{\ell(\psi)},$$

donc ρ est un unificateur de ψ_σ et que $\tau \in \{ \rho \circ \sigma \mid \rho \in \text{Unif}[\psi_\sigma] \}$.

Nous avons montré que l'ensemble sur la gauche de (2.4) est inclus dans celui de droite. Montrons donc l'autre inclusion. Si $\rho \in \text{Unif}[\psi_\sigma]$, alors

$$\begin{aligned} \psi_{\rho \circ \sigma} &= (\psi_\sigma)_\rho \in \Delta^{\ell(\psi_\sigma)} = \Delta^{\ell(\psi)}, & \text{car } \rho \in \text{Unif}[\psi_\sigma] \\ \pi_{\rho \circ \sigma} &= (\pi_\sigma)_\rho \in (\Delta^{\ell(\pi)})_\rho \subseteq \Delta^{\ell(\pi)}, & \text{car } \sigma \in \text{Unif}[\pi] \end{aligned}$$

donc $\rho \circ \sigma \in \text{Unif}[\pi, \psi]$. Ceci complète la preuve de l'égalité (2.4).

Soient maintenant ρ un MGU de ψ_σ et soit $\tau \in \text{Unif}[\pi, \psi]$. Grâce à l'égalité (2.4), nous pouvons écrire $\tau = \rho' \circ \sigma$ avec $\rho' \in \text{Unif}[\psi_\sigma]$; on a alors $\rho \leq \rho'$, donc $\rho' = \theta \circ \rho$ pour une substitution θ et, par conséquent, $\tau = \theta \circ \rho \circ \sigma$, ce qui montre que $\rho \circ \sigma \leq \tau$. Nous avons donc montré que $\rho \circ \sigma$ est un MGU du problème π, ψ . \square

La preuve de correction et complétude de l'algorithme d'unification repose sur les lemmes suivants :

Lemme 2.80. *Les faits suivants sont vrais :*

1. Si $f \neq g$, alors $\text{Unif}[(f(r_1, \dots, r_k), g(r'_1, \dots, r'_{k'}))] = \emptyset$.
2. Si $x \in \text{Var}(t)$ et $t \neq x$, alors $\text{Unif}[(x, t)] = \emptyset$.
3. $\text{Unif}[\pi, \psi] = \text{Unif}[\psi, \pi] \subseteq \text{Unif}[\psi]$.
En particulier, si $\text{Unif}[(s, t)] = \emptyset$, alors $\text{Unif}[(s, t), (s_2, t_2), \dots, (s_n, t_n)] = \emptyset$.

Lemme 2.81. *On a*

$$\begin{aligned} &\text{Unif}[(f(r_1, \dots, r_k), f(r'_1, \dots, r'_k)), (s_2, t_2), \dots, (s_n, t_n)] \\ &= \text{Unif}[(r_1, r'_1), \dots, (r_k, r'_k), (s_2, t_2), \dots, (s_n, t_n)]. \end{aligned}$$

Lemme 2.82. *Un MGU de*

$$(x, x), (s_2, t_2), \dots, (s_n, t_n)$$

est ρ , où ρ est un MGU de

$$(s_2, t_2), \dots, (s_n, t_n).$$

Si ce dernier problème ne possède pas de solution, alors il en est de même pour $(x, x), (s_2, t_2), \dots, (s_n, t_n)$.

Lemme 2.83. *Supposons $x \notin \text{Var}(t)$. Un MGU de*

$$(x, t), (s_2, t_2), \dots, (s_n, t_n)$$

est $\rho \circ [t/x]$, où ρ est un MGU de

$$(s_2, t_2)[t/x], \dots, (s_n, t_n)[t/x].$$

Si ce dernier problème ne possède pas de solution, alors il en est de même pour $(x, t), (s_2, t_2), \dots, (s_n, t_n)$.

- Les Lemmes 2.82 et 2.83 sont des conséquences de la Proposition 2.79, en raison du fait que
- la substitution identité \square est un MGU du problème (x, x) ;
 - $[t/x]$ est évidemment un MGU du problème (x, t) quand $x \notin \text{Var}(t)$.

Exercice 2.84. A l'aide des Lemmes 2.80-2.83 complétez une preuve formelle de correction et complétude de l'algorithme d'unification.

Exemple : démonstration de la Proposition 2.77, correction de l'algorithme. L'algorithme retourne échec, sans appels récursifs, à cause des lignes 4,10 et 13. Pour les lignes 4 et 10, on utilise le Lemme 2.80. Pour la ligne 13, on utilise le Lemme 2.83.

Les appels récursifs se trouvent aux lignes 6 et 9. On justifie la ligne 6 par le Lemme 2.81, et la ligne 9 par le e Lemme 2.82.

Nous argumentons ainsi que si $\text{UNIFIER}(\pi)$ retourne ECHEC, alors $\text{Unif}[(\]\pi) = \emptyset$.

La preuve que si $\text{UNIFIER}(\pi)$ retourne σ , alors σ est un MGU de π est similaire. □

2.7 Résolution

2.7.1 Substitution, sur les formules propositionnelles

L'action d'une substitution s'étend aisément aux formules sans quantificateurs :

- $R(t_1, \dots, t_n)\sigma = R(t_1\sigma, \dots, t_n\sigma)$,
- $(\neg\varphi)\sigma = \neg(\varphi\sigma)$,
- $(\varphi \circ \psi)\sigma = \varphi\sigma \circ \psi\sigma$, $\circ \in \{\vee, \wedge, \Rightarrow\}$.

Rappel (cf. Définition 2.56). Un **littéral** est ou bien une formule atomique, ou bien la négation d'une formule atomique. Une **clause universelle** est la fermeture universelle d'une disjonction de littéraux.

Désormais, clause sera un synonyme de clause universelle. Bien que une clause soit une formule de la forme

$$\forall x_1, \dots, \forall x_n (l_1 \vee \dots \vee l_k)$$

avec l_i des littéraux et $\{x_1, \dots, x_n\} = \text{FV}(\{l_1, \dots, l_n\})$, il est habituel de laisser l'écriture des quantificateurs implicite. Par exemple, nous allons considérer l'expression

$$R(x, y) \vee \neg Q(f(x), z)$$

comme un raccourci de sa fermeture universelle :

$$\forall x \forall y \forall z (R(x, y) \vee \neg Q(f(x), z)).$$

Pour de raison de convenance, nous avons donc décidé d'écrire de la même façon une clause universelle et sa matrice (la sous-formule sans quantificateurs). Au cas nous aurions besoin de distinguer une clause universelle C de sa matrice, nous allons écrire C_{mat} pour la matrice.

La substitution s'étend, en particulier, aux littéraux et aux clauses :

1. $R(t_1, \dots, t_n)\sigma = R(t_1\sigma, \dots, t_n\sigma)$,
2. $(\neg R(t_1, \dots, t_n))\sigma = \neg(R(t_1, \dots, t_n)\sigma)$,
3. $(\bigvee_{i=1}^n l_i)\sigma = \bigvee_{i=1}^n (l_i)\sigma$.

Notation 2.85. Si C est une clause universelle et σ une substitution, nous allons utiliser la notation $C\sigma$ pour la fermeture universelle de $C_{\text{mat}}\sigma$. Évidemment, $C\sigma$ est aussi une clause universelle.

Un **unificateur** de deux littéraux l_0 et l_1 est une substitution σ telle que $l_0\sigma = l_1\sigma$.

Lemme 2.86. σ est un unificateur de l_0 et l_1 ssi

1. $l_0 = R(s_1, \dots, s_n)$, $l_1 = R(t_1, \dots, t_n)$, et $\sigma \in \text{Unif}[(s_1, t_1), \dots, (s_n, t_n)]$, ou bien
2. $l_0 = \neg R(s_1, \dots, s_n)$, $l_1 = \neg R(t_1, \dots, t_n)$, et $\sigma \in \text{Unif}[(s_1, t_1), \dots, (s_n, t_n)]$.

Du Lemme il en découle tout de suite que l'ensemble des unificateurs de deux littéraux, appelons encore une fois $\text{Unif}[l_0, l_1]$, ou bien il est vide, ou bien il possède une **substitution la plus générale**, qui sera appelé un MGU de l_0 et l_1 .

2.7.2 Les règles du calcul de la résolution

On peut considérer le calcul de la résolution comme une généralisation de la méthode de la coupure propositionnelle. Comme dans le cas propositionnel, la méthode de résolution prend en paramètre un ensemble Γ de clauses (universelles) et essaye de dériver la clause vide \perp depuis les clauses dans Γ . Les deux règles pour dériver des nouvelles clauses à partir des clauses déjà construites sont les suivantes :

$$\frac{C \vee A_0 \quad C' \vee \neg A_1}{(C \vee C')\sigma} \text{ Résolution}$$

où σ est un MGU des formules atomiques A_0 et A_1 , et

$$\frac{C \vee l_0 \vee l_1}{(C \vee l_0)\sigma} \text{ Factorisation}$$

où σ est un MGU des littéraux l_0 et l_1 .

Exemple 2.87 (Règle de Résolution). La suivante est une instance de la règle de résolution :

$$\frac{\neg \text{HabiteCL}(x) \vee \text{Tue}(x, \text{Agate}) \quad \neg \text{Tue}(\text{Charles}, y) \vee \text{Hait}(x, y)}{\neg \text{HabiteCL}(\text{Charles}) \vee \text{Hait}(\text{Charles}, \text{Agate})}$$

Ici $l_0 = \text{Tue}(x, \text{Agate})$ et $l_1 = \text{Tue}(\text{Charles}, y)$, $C = \text{HabiteCL}(x)$, $C' = \text{Hait}(x, y)$. En fait, $[\text{Charles}/x, \text{Agate}/y]$ est un MGU de $\text{Tue}(x, \text{Agate})$ et $\text{Tue}(\text{Charles}, y)$.

Exemple 2.88 (Règle de Factorisation). La suivante est une instance de la règle de factorisation :

$$\frac{\neg \text{HabiteCL}(x) \vee \text{Hait}(x, y) \vee \text{Tue}(x, \text{Agate}) \vee \text{Tue}(\text{Charles}, y)}{\neg \text{HabiteCL}(\text{Charles}) \vee \text{Hait}(\text{Charles}, \text{Agate}) \vee \text{Tue}(\text{Charles}, \text{Agate})}$$

Le MGU est encore une fois $[\text{Charles}/x, \text{Agate}/y]$.

2.7.3 Correction du calcul de la résolution

La méthode de résolution est correcte, c'est-à-dire, la conclusion d'une règle est conséquence logique des prémisses de la règle. Explicitement, chaque fois que $\mathcal{M} \models C_i$, où C_i , $i = 1, \dots, n$ avec $n \in \{1, 2\}$ sont les prémisses d'une règle, alors on a $\mathcal{M} \models C_0$, où C_0 est la conclusion de la règle.

En fait, on peut penser que les règles du calcul sont obtenues comme synthèse de deux règles, l'une qui porte sur les substitutions (et les quantificateurs universels), et l'autre étant la règle correspondante propositionnelle :

$$\frac{\frac{C \vee A_0}{C\sigma \vee A_0\sigma} \sigma \quad \frac{C' \vee \neg A_1}{C'\sigma \vee \neg A_0\sigma} \sigma}{C\sigma \vee C'\sigma} \text{ Résolution propositionnelle}$$

où on a $A_0\sigma = A_1\sigma$. De façon semblable :

$$\frac{\frac{C \vee l_0 \vee l_1}{C\sigma \vee l_0\sigma \vee l_1\sigma} \sigma}{C\sigma \vee l_0\sigma} \text{ Factorisation propositionnelle}$$

Nous allons, dans la suite, justifier ces règles "plus élémentaires".

Lemme 2.89. Soient φ une formule sans quantificateurs, σ une substitution, \mathcal{M} une \mathcal{S} -structure et \mathcal{V} une valuation. On a que

$$\mathcal{M}, \mathcal{V} \models \varphi \text{ ssi } \mathcal{M}, \mathcal{V}\sigma \models \varphi, \quad (2.5)$$

où $\mathcal{V}\sigma$ est la valuation définie par la règle suivante :

$$(\mathcal{V}\sigma)(x) = \llbracket \sigma(x) \rrbracket_{\mathcal{M}, \mathcal{V}}.$$

Notez que si $\sigma = [t_1/x_1, \dots, t_n/x_n]$, alors $\mathcal{V}\sigma = \mathcal{V}[x_1 := \llbracket t_1 \rrbracket_{\mathcal{M}, \mathcal{V}}, \dots, x_n := \llbracket t_n \rrbracket_{\mathcal{M}, \mathcal{V}}]$ est la variante de \mathcal{V} satisfaisant aux lois suivantes :

$$\mathcal{V}[x_1 := \llbracket t_1 \rrbracket_{\mathcal{M}, \mathcal{V}}, \dots, x_n := \llbracket t_n \rrbracket_{\mathcal{M}, \mathcal{V}}](y) = \begin{cases} \llbracket t_i \rrbracket_{\mathcal{M}, \mathcal{V}}, & \text{si } y = x_i, \text{ pour quelques } i, \\ \mathcal{V}(y), & \text{sinon.} \end{cases}$$

Démonstration du Lemme 2.89. La preuve de cet énoncé se fait aisément par induction. Nous nous limiterons à illustrer le cas de base. Pour toute variable $y \in X$, nous avons

$$\llbracket y \rrbracket_{\mathcal{M}, \mathcal{V}\sigma} = (\mathcal{V}\sigma)(y) = \llbracket \sigma(y) \rrbracket_{\mathcal{M}, \mathcal{V}}.$$

Ainsi, nous avons

$$\begin{aligned} \mathcal{M}, \mathcal{V} \models R(y_1, \dots, y_m)\sigma &\text{ ssi } \mathcal{M}, \mathcal{V} \models R(\sigma(y_1), \dots, \sigma(y_m)) \\ &\text{ ssi } (\llbracket \sigma(y_1) \rrbracket_{\mathcal{M}, \mathcal{V}}, \dots, \llbracket \sigma(y_m) \rrbracket_{\mathcal{M}, \mathcal{V}}) \in R^{\mathcal{M}} \\ &\text{ ssi } (\llbracket y_1 \rrbracket_{\mathcal{M}, \mathcal{V}\sigma}, \dots, \llbracket y_m \rrbracket_{\mathcal{M}, \mathcal{V}\sigma}) \in R^{\mathcal{M}} \\ &\text{ ssi } \mathcal{M}, \mathcal{V}\sigma \models R(y_1, \dots, y_m). \quad \square \end{aligned}$$

Lemme 2.90. *Pour toute clause universelle C et toute substitution σ , la règle d'inférence suivante est correcte :*

$$\frac{C}{C\sigma}$$

C'est-à-dire, si $\mathcal{M} \models C$, alors $\mathcal{M} \models C\sigma$, pour toute \mathcal{S} -structure \mathcal{M} .

Démonstration. Supposons que $\mathcal{M} \models C$; pour montrer que $\mathcal{M} \models C\sigma$, nous devons montrer que $\mathcal{M}, \mathcal{V} \models (C\sigma)_{\text{mat}}$, où \mathcal{V} est une valuation arbitraire. En considération que $(C\sigma)_{\text{mat}} = (C_{\text{mat}})\sigma$ et par le Lemme 2.89 cela revient à vérifier que $\mathcal{M}, \mathcal{V}\sigma \models C_{\text{mat}}$; cette dernière relation est en effet vraie à cause de l'assomption $\mathcal{M} \models C$. \square

Pour les règles de factorisation et résolution propositionnelles, nous devons les justifier. En fait, ces règles manipulent des clauses universelles et non pas simplement des clauses. Par ailleurs, les démonstrations que ces règles propositionnelles s'étendent au cas des clauses universelles sont assez faciles.

Lemme 2.91. *Pour toute couple de clauses universelles de la forme $C_1 \vee l$ et $C_2 \vee \neg l$ (avec l un littéral), la règle d'inférence suivante est correcte :*

$$\frac{C_1 \vee l \quad C_2 \vee \neg l}{C_1 \vee C_2} \text{ Résolution propositionnelle}$$

C'est-à-dire, si $\mathcal{M} \models C_1 \vee l$ et $\mathcal{M} \models C_2 \vee \neg l$, alors $\mathcal{M} \models C_1 \vee C_2$, pour toute \mathcal{S} -structure \mathcal{M} .

Démonstration. Nous devons montrer que $\mathcal{M}, \mathcal{V} \models (C_1 \vee C_2)_{\text{mat}}$ pour toute valuation \mathcal{V} . Cela est une conséquence de $(C_1 \vee C_2)_{\text{mat}} = (C_1)_{\text{mat}} \vee (C_2)_{\text{mat}}$, du fait que $\mathcal{M}, \mathcal{V} \models (C_1 \vee l)_{\text{mat}} (= (C_1)_{\text{mat}} \vee l)$, $\mathcal{M}, \mathcal{V} \models (C_2 \vee \neg l)_{\text{mat}} (= (C_2)_{\text{mat}} \vee \neg l)$, et du fait que la règle de la coupure propositionnelle est correcte. \square

Exercice 2.92. Montrez que la règle de factorisation propositionnelle s'étend aux clauses universelles.

2.7.4 Complétude du calcul de la résolution

Soit Γ un ensemble de clauses universelles. Nous disons que Γ est **saturé** si toute clause dérivable (via résolution et factorisation) de formules dans Γ appartient déjà à Γ ; nous disons que Γ est **cohérent** si $\perp \notin \Gamma$.

Théorème 2.93. *Un ensemble saturé et cohérent de clauses admet au moins un modèle.*

En fait, nous allons montrer la proposition suivante :

Proposition 2.94. *Si un ensemble saturé de clauses Γ n'admet pas un modèle, alors $\perp \in \Gamma$.*

Démonstration. Soit $\mathcal{S} = (\mathcal{S}_f, \mathcal{S}_r)$ le langage; nous allons nous intéresser au langage propositionnel caractérisé par le fait que PROP est l'ensemble de proposition atomiques du langage \mathcal{S} .

Remarquons d'abord que, étant donnée une valuation (au sens propositionnel) $v : \text{PROP} \rightarrow \{0, 1\}$, nous pouvons définir une \mathcal{S} -structure \mathcal{M}_v de la façon suivante :

- $D_{\mathcal{M}_v} := \mathcal{T}_{\mathcal{S}_f}(X)$;
- pour tout $f \in \mathcal{S}_f$, $f^{\mathcal{M}_v}$ est la fonction qui envoie un tuple $(t_1, \dots, t_n) \in \mathcal{T}_{\mathcal{S}_f}(X)$ vers le terme $f(t_1, \dots, t_n)$;
- pour tout $R \in \mathcal{S}_r$, nous posons

$$R^{\mathcal{M}_v} := \{ (t_1, \dots, t_n) \in \mathcal{T}_{\mathcal{S}_f}(X)^n \mid v(R(t_1, \dots, t_n)) = 1 \}.$$

La structure \mathcal{M}_v ainsi définie a cette propriété importante :

Lemme 2.95. *Si C est une clause universelle, alors*

$$\mathcal{M}_v \models C \text{ ssi } v(C\sigma) = 1, \text{ pour toute substitution } \sigma.$$

Preuve du Lemme. Une valuation \mathcal{V} de l'ensemble de variables vers $D_{\mathcal{M}_v} = \mathcal{T}_{\mathcal{S}_f}(X)$ n'est rien d'autre qu'une substitution. En tenant compte que C est implicitement quantifiée universellement, la condition $\mathcal{M}_v \models C$ est vraie quand $\mathcal{M}_v, \sigma \models C_{\text{mat}}$, pour toute substitution σ . Rappelons que $\llbracket \cdot \rrbracket$ est la substitution identité; le Lemme 2.89 montre que $\mathcal{M}_v, \sigma \models C_{\text{mat}}$ est équivalent à $\mathcal{M}_v, \llbracket \cdot \rrbracket \models C_{\text{mat}}\sigma$; cette condition revient à dire que $v(C\sigma) = 1$ (au sens propositionnel). \square

Il en découle que l'ensemble de clauses propositionnelles

$$\Delta = \{ C\sigma \mid C \in \Gamma, \sigma \text{ une substitution} \}$$

n'est pas satisfaisable au sens propositionnel. En fait, si v est une valuation vérifiant toutes les formules de cet ensemble, alors \mathcal{M}_v est un modèle satisfaisant toutes les formules de Γ .

Pour le Théorème de compacité, il existe un sous-ensemble fini $\Delta_f \subseteq \Delta$ tel que Δ_f n'est pas satisfaisable. Car la méthode de la coupure est complète, il existe une suite de clauses D_1, \dots, D_n avec $D_n = \perp$ (c'est-à-dire, D_n est la clause vide), telle que, pour tout $i > 0$:

1. $D_i \in \Delta_f$, ou
2. D_i est déduite de D_j et D_k (avec $j, k < i$) via la règle de coupure, ou
3. D_i est déduite de D_j (avec $j < i$) via la règle de factorisation (propositionnelle).

Lemme 2.96. *Pour tout $i = 1, \dots, n$, ils existent $C_i \in \Gamma$ et une substitution ρ_i telle que $D_i = C_i\rho_i$.*

Preuve du Lemme. Par induction (sur $i = 1, \dots, n$), et par cas.

1. Si $D_i \in \Delta_f \subseteq \Delta$, alors cela est vrai par définition de Δ : $D_i = C \circ \sigma$ pour une clause $C \in \Gamma$ et une substitution σ ; on peut donc poser $C_i := C$ et $\rho_i := \sigma$.
2. (Voir la Figure 2.3.) Supposons que D_i est déduite de D_j et D_k (avec $j, k < i$) via la règle de coupure. Par hypothèse d'induction, ils existent $C_j, C_k \in \Gamma$ et deux substitutions ρ_j, ρ_k tels que $D_j = C_j\rho_j$ et $D_k = C_k\rho_k$.

Supposons donc que $D_j = D \vee A$, $D_k = D' \vee \neg A$, et $D_i = D \vee D'$. On a alors $C_j = C \vee A_0$, $C_k = C' \vee \neg A_1$, $D = C\rho_j$, $D' = C'\rho_k$, et $A_0\rho_j = A = A_1\rho_k$. Sans perte de généralité, nous pouvons assumer qu'il n'y a pas des variables en commun entre C_j et C_k , que ρ_j fixe les variables de C_k , et ρ_k fixe les variables de C_j . Par conséquent, si $\rho_j = [t_1/x_1, \dots, t_n/x_n]$ et $\rho_k = [s_1/y_1, \dots, s_m/y_m]$, alors $\tau = [t_1/x_1, \dots, t_n/x_n, s_1/y_1, \dots, s_m/y_m]$ est un unificateur de A_0 et A_1 , $D_j = C_j\tau$ et $D_k = C_k\tau$. Soit σ un MGU de A_0 et A_1 , on a alors $\tau = \rho \circ \sigma$, et

$$D_i = D \vee D' = C\tau \vee C'\tau = (C \vee C')\tau = [(C \vee C')\sigma]\rho.$$

Nous pouvons alors poser $C_i := (C \vee C')\sigma$ et $\rho_i := \rho$. $C_i \in \Gamma$ car elle est déduite de C_i et C_j via la règle de résolution depuis $C_j, C_k \in \Gamma$, et en plus Γ est saturé.

3. Supposons enfin que la clause D_i est déduite de D_j (avec $j < i$) via la règle de factorisation propositionnelle. Par hypothèse d'induction, il existe une clause $C_j \in \Gamma$ et une substitution ρ_j telle que $D_j = C_j\rho_j$.
Si $D_j = D \vee \ell \vee \ell$, alors $C_j = C \vee \ell_0 \vee \ell_1$ avec $D = C\rho_j$, et $\ell = \ell_0\rho_j = \ell_1\rho_j$. La substitution ρ_j est donc un unificateur de ℓ_0 et ℓ_1 ; si σ est un MGU de ℓ_0 et ℓ_1 , alors il existe une substitution ρ telle que $\rho_j = \rho \circ \sigma$.

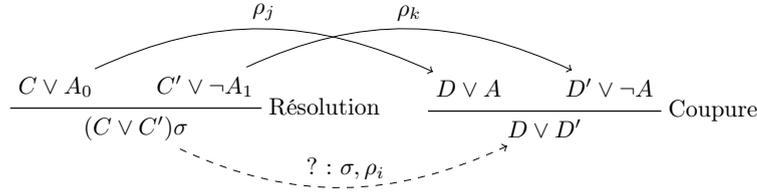


FIGURE 2.3 – Simulation de la coupure par la résolution

Nous pouvons donc poser $C_i := (C \vee \ell_0)\sigma$ et $\rho_i := \rho$; $C_i \in \Gamma$ car il a été déduit depuis $C_j \in \Gamma$ via la règle de factorisation (du premier ordre) et Γ saturé; par ailleurs

$$C_i \rho_i = [(C \vee \ell_0)\sigma] \rho_i = (C \vee \ell_0)(\rho_i \circ \sigma) = (C \vee \ell_0)\rho_j = C \rho_j \vee \ell_0 \rho_j = D \vee \ell = D_j. \quad \square$$

En particulier, le Lemme dit que $C_n \in \Gamma$ et que $C_n \rho_n = \perp$. Il est facile à voir que si $C_n \neq \perp$, alors $C_n \rho \neq \perp$ (pour n'importe quel ρ); par conséquent, nous avons $C_n = \perp \in \Gamma$. Cela complète la démonstration de la Proposition 2.94. \square

Une analyse fine de la preuve de la Proposition 2.94 amène à une preuve du théorème suivant.

Théorème 2.97. *Si un ensemble de clauses Γ n'admet pas un modèle, alors il existe une preuve de \perp à partir de Γ dans le calcul de la résolution.*

Exemple 2.98. L'ensemble de clauses universelles Γ défini par

$$\Gamma := \{ P(x) \vee Q(f(y)), \neg P(c), \neg Q(f(g(x))) \vee Q(g(x)), \neg Q(f(g(x))) \vee Q(g(x)), \neg Q(g(c)) \},$$

est insatisfaisable. A l'aide de la méthode de la coupure, on peut donc construire une preuve de la clause vide à partir de l'ensemble $\Delta = \{ C\sigma \mid C \in \Gamma, \sigma \text{ une substitution} \}$. Cette preuve peut ensuite être simulée, par le Lemme 2.96, dans le calcul de la résolution, pour produire une preuve de la clause vide dans ce calcul. La figure 2.4 montre une preuve par coupure de la clause vide de Δ et sa simulation par résolution depuis Γ .

$$\frac{\frac{\frac{P(c) \vee Q(f(g(c))) \quad \neg Q(f(g(c))) \vee Q(g(c))}{P(c) \vee Q(g(c))} \text{ Coupure} \quad \neg P(c)}{Q(g(c))} \text{ Coupure} \quad \neg Q(g(c))}{\perp} \text{ Coupure}$$

$$\frac{\frac{\frac{P(x) \vee Q(f(y)) \quad \neg Q(f(g(x))) \vee Q(g(x))}{P(x) \vee Q(g(x))} \text{ Résolution } y \mapsto g(x) \quad \neg P(c)}{Q(g(c))} \text{ Résolution } x \mapsto c \quad \neg Q(g(c))}{\perp} \text{ Résolution}$$

FIGURE 2.4 – Simulation de preuves, exemple

2.7.5 Indécidabilité

Bien que le calcul soit correct et complet, nos résultats n'amènent pas à la construction d'un algorithme—c'est à un quelque programme qui *s'arrête toujours* et qui donne la réponse souhaitée à la fin des calculs—pour décider si un ensemble de clauses universelles est satisfaisable ou non. Si nous essayons d'adapter l'algorithme de résolution propositionnelle, cf. ??, on rencontre un problème majeur : cet algorithme pourrait ne jamais se terminer, en raison de la possibilité de

produire une infinité de nouvelles de clauses. Pour s'en apercevoir, il suffit de considérer le langage \mathcal{S} avec $\mathcal{S}_F = \{(o, 0), (s, 1)\}$ et $\mathcal{S}_R = \{(P, 1)\}$. Considérons l'ensemble \mathcal{C} de clauses donné par

$$\mathcal{C} := \{\neg P(x) \vee P(s(x)), P(o)\}.$$

L'algorithme engendrera, l'une après l'autres, toutes les clauses de la forme

$$P(\underbrace{s(\dots s(o)\dots)}_{n \text{ fois}})$$

En fait, nous ne pouvons simplement pas trouver un algorithme ; les prochains théorèmes pourront être mieux compris dans le cadre du chapitre suivant, autour de la calculabilité, où nous formaliserons la notion d'algorithme.

Théorème 2.99. *Il n'existe aucun algorithme tel que, étant donné une formule du premier ordre close φ , il répond oui si φ admet un modèle, et non si φ est insatisfaisable.*

Puisque décider de la satisfaisabilité d'une formule du premier ordre se réduit (via la mise en forme clausale) à décider de la satisfaisabilité d'un ensemble de clauses, nous pouvons déduire cet autre théorème à partir du précédent :

Théorème 2.100. *Il n'existe aucun algorithme qui, étant donné un ensemble fini de clauses \mathcal{C} , répond oui si \mathcal{C} admet un modèle, et non si \mathcal{C} est insatisfaisable.*

2.7.6 Utilisation d'un démonstrateur automatique

Exemple 2.101 (L'île mystérieuse). Écoutez cette histoire.

Chaque habitant de cette île est soit un cavaliers, soit un escroc. Il peut être un loup garou (il est donc dangereux, car il mange les hommes pendant les nuits de lune pleine). Un loup garou est lui aussi soit un cavalier soit un escroc. Les cavaliers disent toujours la vérité, les escrocs mentent toujours. Un explorateur débarque sur cette île et rencontre Albert, Bernard et Charles. Il est au courant qu'un des trois est un loup garou.

- Albert prétend que Bernard est un loup ;
- Bernard dit qu'il n'est pas un loup ;
- Charles dit qu'au moins deux entre eux sont des escrocs.

Qui doit choisir l'explorateur comme guide de son voyage ?

Nous avons formalisé cette histoire en logique du premier ordre, dans un fichier prêt à être lu par le démonstrateur automatique **Prover9**. Ce fichier apparaît dans la Figure 2.5. Le prouveur automatique confirme que Albert est un loup garou, et donc l'explorateur ne choisira pas Albert comme guide. La preuve construite par le prouveur apparaît dans la Figure 2.6. Le lecteur y reconnaîtra plusieurs instances de la règle de résolution. L'analyse de la preuve montre que l'hypothèse 'Albert n'est pas un loup garou' n'a pas été utilisée. Cela veut dire que la connaissances à disposition de l'explorateur, (qui est modélisée dans la liste des assumptions) est elle même incohérente.

Un procédé analogue peut être utilisé pour montrer que une liste de spécifications d'un programme/logiciel est incohérente, et donc ne peut pas être assuré par n'importe quel programme. Réfléchir avant se mettre à programmer !!! □

Exemple 2.102 (Le mystère du Château Letot). Écoutez cette autre histoire.

```

set(binary_resolution).

formulas(assumptions).
    Cavalier(x) | Escroc(x).
    LoupGarou(albert) | LoupGarou(bernard) | LoupGarou(charles).
    Cavalier(albert) -> LoupGarou(bernard).
    Escroc(albert) -> -LoupGarou(bernard).
    Cavalier(bernard) -> -LoupGarou(bernard).
    Escroc(bernard) -> LoupGarou(bernard).
    Cavalier(charles) -> (
        (Escroc(albert) & Escroc(bernard))
        | (Escroc(albert) & Escroc(charles))
        | (Escroc(bernard) & Escroc(charles))
    ).
    Escroc(charles) -> -(
        (Escroc(albert) & Escroc(bernard))
        | (Escroc(albert) & Escroc(charles))
        | (Escroc(bernard) & Escroc(charles))
    ).

end_of_list.

formulas(goals).
    LoupGarou(albert).

end_of_list.

```

FIGURE 2.5 – Fichier d’entrée pour Prover9

- *Quelqu’un qui habite Château Letot a tué tante Agate.*
- *Agate, le majordome, et Charles habitent Château Letot, et ils sont les seuls qui l’habitent.*
- *Un tueur haït toujours sa victime, et il n’est jamais plus riche que sa victime.*
- *Charles haït personne que tante Agate haït.*
- *Agate haït tous sauf le majordome.*
- *Le majordome haït tous ceux qui ne sont pas plus riches de tante Agate.*
- *Le majordome haït tous ceux que tante Agate haït.*
- *Personne haït tous le monde.*
- *Agate n’est pas le majordome.*

Qui a tué tante Agate ?

Nous avons utilisé le prouveur automatique **Prover9** pour montrer que Agate s’est suicidé ; en fait, ‘Agate a tué Agate’ est une conséquence logique des faits décrits concernant le Château Letot.

Le procédé est comme auparavant (voir l’île mystérieuse). Nous avons d’abord modélisé l’histoire (*base de connaissances*, ou *ontologie*) en logique du premier ordre, en construisant ainsi un ensemble d’*assumptions* ; la phrase ‘Agate a tué Agate’ étant la formule *but* à démontrer⁴. Le code qui a été fourni en entrée à **Prover9** apparaît dans la figure 2.7. **Prover9** transforme d’abord cet ensemble de formules dans un ensemble de clauses universelles. Observez donc l’introduction de nouvelles constantes et de symboles de fonction par élimination de quantificateurs existentiels (Skolemization), la mise sous forme clausale, et l’inclusion du but parmi les *assumptions*, via sa négation (voir figure 2.8). La preuve que le but est une conséquence logique des *assumptions* apparaît dans la figure 2.9 Nous avons utilisé les outils GVIZIFY (pour transformer la sortie du **Prover9**—très souvent assez difficile à décrypter—vers un graphe décrit dans le langage *dot*) est GRAPHVIZ (pour dessiner des graphes à partir de leur description en langage *dot*) afin de présenter cette preuve sous forme de diagramme.

Une dernière remarque s’impose. Parmi les symboles de relation de notre langage nous avons utilisé le symbole = sans avoir ajouté, parmi les *assumptions*, aucune hypothèse sur ce symbole.

4. Dans la notation que nous avons utilisée dans le cours, les *assumptions* correspondent à l’ensemble Γ et le but à φ quand on se pose la question si φ est une conséquence logique de Γ ($\Gamma \models \varphi$?)

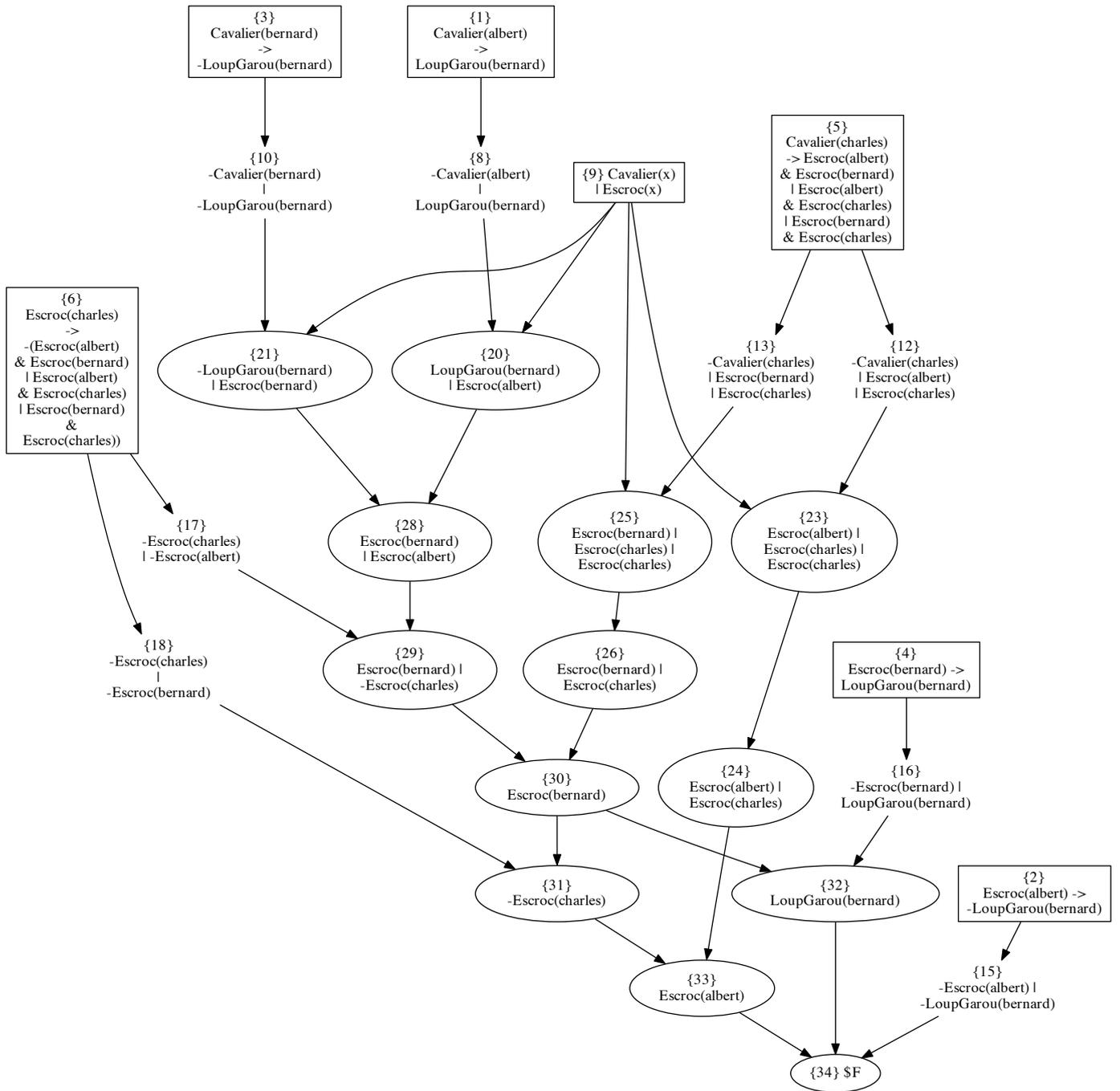


FIGURE 2.6 – Visualisation de la preuve construite par Prover9

```

set(binary_resolution).

formulas(assumptions).
    exists x (HabiteCL(x) & Tue(x,agate)).
    HabiteCL(agate) & HabiteCL(maj) & HabiteCL(charles) &
        (HabiteCL(x) -> (x = agate | x = maj | x = charles)).
    Tue(x,y) -> (Hait(x,y) & -PlusRiche(x,y)).
    Hait(agate,z) -> -Hait(charles,z).
    (-Hait(agate,x)) -> x=maj.
    x != maj -> Hait(agate,x).
    -PlusRiche(x,agate) -> Hait(maj,x).
    Hait(agate,x) -> Hait(maj,x).
    - (exists x all y Hait(x,y)).
    agate != maj.
end_of_list.

formulas(goals).
    Tue(agate,agate).
end_of_list.

```

FIGURE 2.7 – Fichier entrée pour le château Letot

```

HabiteCL(c1). [clausify(1)].
Tue(c1,agate). [clausify(1)].
HabiteCL(agate). [clausify(2)].
HabiteCL(maj). [clausify(2)].
HabiteCL(charles). [clausify(2)].
-HabiteCL(x) | agate = x | maj = x | charles = x. [clausify(2)].
-Tue(x,y) | Hait(x,y). [clausify(3)].
-Tue(x,y) | -PlusRiche(x,y). [clausify(3)].
-Hait(agate,x) | -Hait(charles,x). [clausify(4)].
Hait(agate,x) | maj = x. [clausify(5)].
maj = x | Hait(agate,x). [clausify(6)].
PlusRiche(x,agate) | Hait(maj,x). [clausify(7)].
-Hait(agate,x) | Hait(maj,x). [clausify(8)].
-Hait(x,f1(x)). [clausify(9)].
agate != maj. [assumption].
-Tue(agate,agate). [deny(10)].

```

FIGURE 2.8 – Château Letot : forme clause

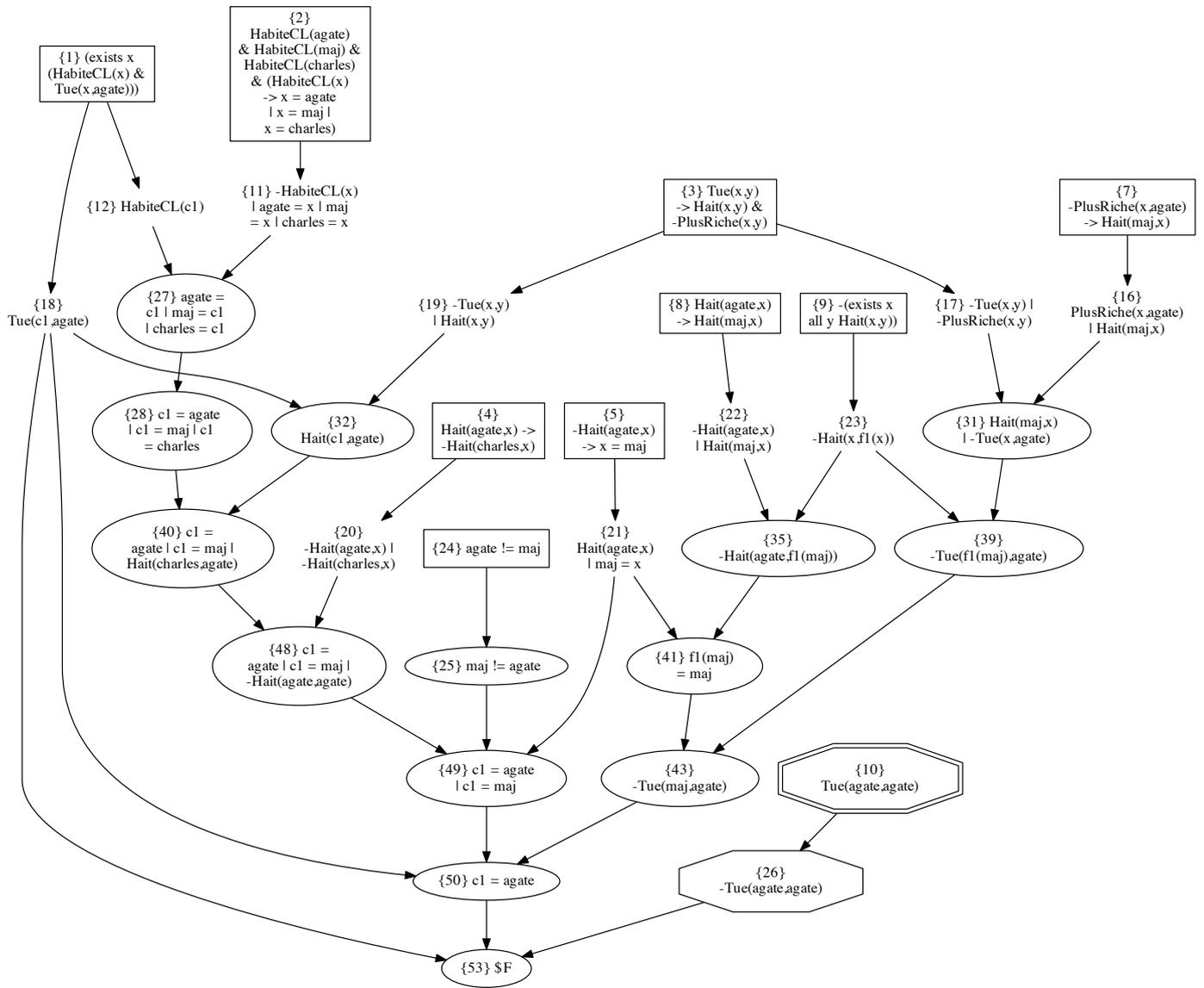


FIGURE 2.9 – La preuve trouvée par Prover9

Notamment, nous aurions du ajouter des clauses explicitant le fait que l'égalité est réflexive, transitive, symétrique, et congruentielle. Par exemple, nous aurions du expliciter que si $x = y$ et $Hait(x, z)$ alors $Hait(y, z)$, et tous les inférences de ce type. **Prover9** reconnaît qu'il s'agit du symbole d'égalité et ajoute ces assumptions automatiquement. Car le traitement de l'égalité n'est pas optimal en utilisant la résolution seulement, on se sert aussi de la règle de paramodulation, que nous présentons ci-dessous.

$$\frac{C \vee t_1 = t_2 \quad D(t_3)}{(C \vee D(t_2))\sigma} \text{Paramodulation}$$

où σ est un unificateur de t_1 et t_3 .

Dans notre exemple, nous avons que l'inférence de la clause 40,

$$\frac{c1 = agate \vee c1 = maj \vee c1 = charles \quad Hait(c1, agate)}{c1 = agate \vee c1 = maj \vee Hait(charles, agate)}$$

est une instance de la règle de paramodulation. □

Bibliographie

- [Miq05] Alexandre Miquel. L'intuitionnisme : où l'on construit une preuve. *Pour la Science*, (49), 2005. http://www.dossierpurlascience.fr/ewb_pages/f/fiche-article-1-intuitionnisme-ou-l-on-construit-une-preuve-21944.php.