Cours Logique et Calculabilité

L3 Informatique 2014/2015

Texte par Séverine Fratani, avec addenda par Luigi Santocanale
Version du 31 mars 2015

Table des matières

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

2 Calcul propositionnel **7**
2.1 Introduction 7
2.2 Syntaxe du calcul propositionnel : les formules 7
2.3 Sémantique du calcul propositionnel 8
2.3.1 Modèles d'une formule 10
2.3.2 La conséquence logique (d'un ensemble de formules) 12
2.3.3 Décidabilité du calcul propositionnel 17
2.4 Equivalence entre formules 18
2.4.1 La substitution 18
2.4.2 Equivalences classiques 18
2.4.3 Formes normales 19
2.5 Le problème SAT 22
2.5.1 Définition du problème 22
2.5.2 Un problème NP-complet 22
2.5.3 Modélisation - Réduction à SAT 23
2.5.4 Algorithmes de résolution de SAT 25
2.5.5 Sous-classes de SAT 30
2.5.6 Les SAT-solvers 32
2.5.7 Applications 33
2.5.8 Sur la modélisation 35
2.6 Systèmes de preuves 36
2.6.1 La notion de système formel 36
2.6.2 La méthode de la coupure 36
2.6.3 Systèmes de preuve à la Hilbert 40
2.7 Les règles du calcul des séquents 45
2.8 Résumé 45

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

3.3.5 Interprétation 5 . . . 51
3.3.6 Interprétation 6 . . . 52
3.3.7 Comparaison des interprétations . . . 52
3.4 Expressions et formules . . . 52
3.4.1 Les termes . . . 52
3.4.2 Le langage . . . 53
3.4.3 Les formules du calcul des prédicats . . . 54
3.4.4 Occurrences libres et liées d'une variable . . . 55
3.5 Sémantique . . . 56
3.5.1 Structures . . . 56
3.5.2 Evaluation (des termes et) des formules . . . 58
3.6 Manipulation de formules . . . 63
3.6.1 Substitution de variables . . . 63
3.6.2 Equivalences classiques . . . 63
3.6.3 Formes Normales . . . 64
3.7 Unification . . . 68
3.7.1 Substitions et MGUs . . . 68
3.7.2 Algorithme d'unification . . . 70
3.7.3 Correction et completude . . . 72
3.8 Résolution . . . 74
3.8.1 Substitution, sur les formules propositionnelles . . . 74
3.8.2 Les règles du calcul de la résolution . . . 74
3.8.3 Correction du calcul de la résolution . . . 75
3.8.4 Complètude du calcul de la résolution . . . 76
3.8.5 Indecidabilité . . . 78
3.8.6 Utilisation d'un démonstrateur automatique . . . 79

4 Calculabilité **87**
4.1 Introduction . . . 87
4.2 Machines de Turing . . . 88
4.3 Problèmes de décision . . . 90
4.4 Un problème indécidable . . . 90
4.5 Fonctions récursives . . . 92
4.5.1 Les fonctions primitives recursives . . . 92
4.5.2 Fonctions μ-récursives . . . 95

Chapitre 1

Introduction

1.1 Pourquoi la logique ?

1.1.1 La formalisation du langage

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

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

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

1.1.2 La formalisation du raisonnement

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

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

1.2 Logique et informatique

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

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

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

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

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

1.3 Contenu du cours

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

Nous allons nous intéresser aux fragment suivants :

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

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

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

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

Chapitre 2

Calcul propositionnel

2.1 Introduction

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

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

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

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

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

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

2.2 Syntaxe du calcul propositionnel : les formules

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

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

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

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

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

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

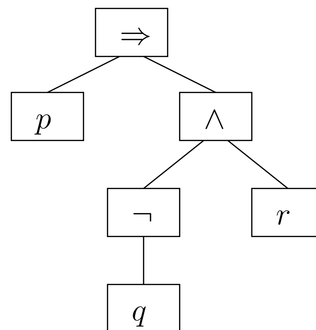


FIGURE 2.1 – Représentation arborescente de la formule $p \Rightarrow (\neg q \wedge r)$

Notation 2.3. On utilise souvent en plus le connecteur binaire $\Leftrightarrow$ comme abréviation : $\varphi \Leftrightarrow \psi$ est l'abréviation de $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$. De la même façon, on ajoute le symbole $\perp$ 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 2.4 (Sous-formule)**.** L'ensemble $SF(\varphi)$ des sous-formules d'une formules φ est défini par induction de la façon suivante.
— $SF(p) = \{p\}$;
— $SF(\neg\varphi) = \{\neg\varphi\} \cup SF(\varphi)$;
— $SF(\varphi \circ \psi) = \{\varphi \circ \psi\} \cup SF(\varphi) \cup SF(\psi)$ (où $\circ$ désigne un des symboles $\wedge$, $\vee$, $\Rightarrow$).

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

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

2.3 Sémantique du calcul propositionnel

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

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

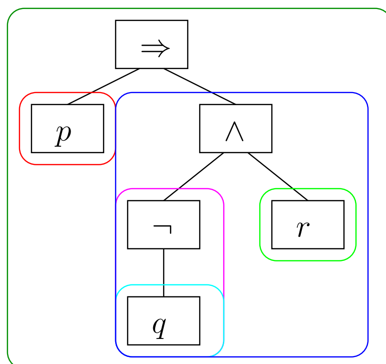


FIGURE 2.2 – Représentation arborescente des sous-formules de $p \Rightarrow (\neg q \wedge r)$

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

p	q
1	0

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

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

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

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

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

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

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

p	$\neg p$
0	1
1	0

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

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

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

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

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

D'autres exemples où il est difficile de coder le langage naturel via le langage formel :
- comment coderiez vous, en langage formel, l'énoncé français « Soit il est frais, soit il est chaud » ?
- et comment coderiez vous l'énoncé anglais « Either I cannot understand French, or my professor doesn't know how to speak it » ?

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

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

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

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

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 $v(\varphi) = v'(\varphi)$.

2.3.1 Modèles d'une formule

Définition 2.12. L'ensemble des **valuations** d'un ensemble de variables propositionnelles Prop est noté $\mathsf{Val}(\text{Prop})$ (ou juste Val lorsqu'il n'y a pas d'ambiguité sur Prop). $\mathsf{Val}(\text{Prop})$ 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 2.3.1, dans lequel 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 2.1 – L'ensemble $\mathsf{Val}(\text{Prop})$ des valuations de $\text{Prop} = \{p, q, r\}$

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

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

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

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

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

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

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

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

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

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

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

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

Exercice 2.21. Prouvez les équivalences suivantes :

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

La proposition suivante explicite la relation intime existante entre logique et algèbre ensembliste. Par la sémantique, nous sommes en fait en train d'étudier et comprendre la logique par ce type d'algèbre (dont nous supposons disposer d'une compréhension intuitive et exacte).

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

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

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

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

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

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

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

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

4.

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

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

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

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

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

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

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

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

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

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

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

On étend les définitions vues précédemment aux ensembles de formules.

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

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

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

— **satisfaisable** (ou **consistant**, ou **cohérent**) s'il admet au moins un modèle (*i.e.*, si $\mathsf{mod}(\Gamma) \neq \varnothing$) ;
— **insatisfaisable** (ou **contradictoire**, ou **inconsistant**, ou encore **incohérent**) s'il n'admet aucun modèle (*i.e.*, si $\mathsf{mod}(\Gamma) = \varnothing$), on note alors $\Gamma \models \perp$.

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

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

Remarque 2.29. Attention aux deux notations :
— $\psi \models \varphi$ où ψ est une formule ;
— $\Gamma \models \varphi$ où Γ est un ensemble de formule.
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 2.30. $\Gamma \models \varphi$ *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 2.31. *Pour tous ensembles de formules* Σ, Γ,

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

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

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

Démonstration de la Proposition 2.31. Pour toute valuation v :

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

□

La preuve de la Proposition suivante est laissée en exercice.

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

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

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

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

Cette proposition exprime le fait qu'un ensemble **fini** de formules propositionnelles peut toujours être vu comme une seule formule formée de la conjonction des formules de l'ensemble. Une formule étant un objet fini, ce résultat ne 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 2.39) qui permet de ramener les problèmes de satisfaisabilité et de contradiction d'un ensemble de taille quelconque à celle d'ensemble finis.

Démonstration. Remarquons que

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

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

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

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

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

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

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

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

On dispose de 4 variables propositionnelles, p,q,r,s, qui obéissent aux contraintes suivantes :

φ_1 : $q\wedge\neg r$
φ_2 : $p\Rightarrow(r\vee s)$
φ_3 : $\neg r\wedge(q\vee p)$

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, $\mathsf{mod}(\varphi_1)\subseteq\mathsf{mod}(\varphi_3)$.
   (b) Soit $\Gamma_2=\{\varphi_1,\varphi_2\}$, on a $\mathsf{mod}(\Gamma_2)=\mathsf{mod}(\Gamma_1)$. En effet, par définition,

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

   Donc les conséquences de Γ_1 et Γ_2 sont les mêmes et on peut alors simplifier Γ_1 par Γ_2.

2. *Quel est l'ensemble des modèles de Γ_2 ?*

Modèles de φ_1 :

p	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'intersections des modèles de φ_1 et φ_2 :

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

(Évidemment, nous aurions pu éviter de calculer tous le 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 :

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

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

2.3.2.1 Compacité

Le théorème de compacité sert à caractériser la conséquence logique dans les cas où l'ensemble des formules est infini en ne considérant que des sous-ensembles finis. Par ailleurs, ce théorème jouera un rôle cruciale plus tard, dans le cadre de la preuve de complètude pour la calcul de la résolution (Théorème 3.94).

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

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

Démonstration. Supposons que T, qui est à branchement fini, soit infini. On définit une branche infinie $e_0, e_1, \ldots, e_n, \ldots$ 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 infinis, ce noeud étant le e_{n+1} cherché. □

Nous aurons besoin du Lemme dans la forme suivante :

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

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

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

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

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

Démonstration (du Théorème 2.40). 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 associé à p_0 : chaque fois que l'on descend d'un noeud de niveau i, ceci revient à poser p_i faux si l'on descend à gauche, et p_i vrai si l'on descend à droite. Remarquons que :

1. chaque chemin infini 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'árbre *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 inconsistant.

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

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

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

Exemple 2.41. Supposons que Γ soit de la forme $\{ p_0 \wedge \neg p_1, p_0 \Rightarrow p_1, \dots \}$. 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 debut de) l'arbre sémantique, avec le noeud 10 étiqueté par la formule témoignant son échec, est représenté en Figure 2.3.

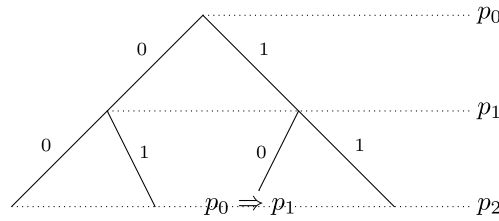


FIGURE 2.3 – Début de l'arbre sémantique avec le noeud d'échec 10 étiqueté

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

Corollaire du théorème de compacité :

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

Démonstration.

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

2.3.3 Décidabilité du calcul propositionnel

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

Théorème 2.44. *Le calcul propositionnel est décidable.*

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

Coût : $O(2^n)$ avec n le cardinal de PROP(φ) (nombre de symboles propositionnels dans φ). □

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

2.4 Equivalence entre formules

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

2.4.1 La substitution

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

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

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

Définition 2.45 (Substitution)**.** Soient φ, ψ, et ψ' trois formules du calcul propositionnel,
- — si ψ n'est pas une sous-formule de φ, alors $\varphi_{[\psi\leftarrow\psi']} = \varphi$
- — sinon si $\varphi = \psi$ alors $\varphi_{[\psi\leftarrow\psi']} = \psi'$
- — sinon
  - — si $\varphi = \neg\varphi'$ alors $\varphi_{[\psi\leftarrow\psi']} = \neg(\varphi'_{[\psi\leftarrow\psi']})$
  - — si $\varphi = \varphi_1 \circ \varphi_2$ (où $\circ$ est un connecteurs $\wedge$, $\vee$, $\Rightarrow$) alors $\varphi_{[\psi\leftarrow\psi']} = \varphi_{1[\psi\leftarrow\psi']} \circ \varphi_{2[\psi\leftarrow\psi']}$.

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

Démonstration. Voir TD 3. □

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

2.4.2 Equivalences classiques

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

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

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

Nous observons aussi que la formule :

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

est une tautologie (et rappelons 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 2.47 (Principle 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 2.48. Nous pouvons argumenter que $\varphi \wedge (\psi \vee \chi) \equiv \varphi \wedge (\chi \vee \psi)$ de façon précise de cette façon :

$$\begin{aligned} \varphi \wedge (\psi \vee \chi) &\equiv (\varphi \wedge p)[p \leftarrow \psi \vee \chi] \\ &\equiv (\varphi \wedge p)[p \leftarrow \chi \vee \psi] \qquad \text{(par commutativité, et en utilisant la proposition 2.46)} \\ &\equiv \varphi \wedge (\chi \vee \psi)\,. \end{aligned}$$

2.4.3 Formes normales

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

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

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

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

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

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

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

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

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

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

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

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

Définition 2.49 (Littéral)**.** Un **littéral** est une formule atomique ou la négation d'une formule atomique. Autrement dit, c'est une formule ℓ de la forme p ou $\neg p$, où p est un symbole propositionnel.

Définition 2.50 (Clause)**.** Une **clause disjonctive** est une disjonction de littéraux : $\bigvee_{i=1}^{n} \ell_i$ où les ℓ_i sont des littéraux. Une **clause conjonctive** est une conjonction de littéraux : $\bigwedge_{i=1}^{n} \ell_i$ où les ℓ_i sont des littéraux.

Remarque 2.51. Comme d'usage, quand le mot « clause » est utilisé sans spécifications ultérieures, l'adjectif « disjonctive » est sous-entendu. Ainsi, *clause* tout-court est synonyme de *clause disjonctive.*

Définition 2.52 (Forme normale conjonctive)**.** Une **formule conjonctive** (ou formule sous **forme normale conjonctive** (FNC), ou sous **forme clausale**) est une conjonction de clauses disjonctives : $\bigwedge_{j=1}^{m} C_j$ où les C_j sont des clauses disjonctives, ou encore $\bigwedge_{j=1}^{m} \bigvee_{i=1}^{n_j} \ell_i^j$ où les ℓ_i^j sont des littéraux.

Exemple 2.53. La formule suivante est sous forme clausale :

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

Définition 2.54 (Forme normale disjonctive)**.** Une **formule disjonctive** (ou formule sous **forme normale disjonctive** (FND)) est une disjonction de clauses conjonctives : $\bigvee_{j=1}^{m} C_j$ où les C_j sont des clauses, ou encore $\bigvee_{j=1}^{m} \bigwedge_{i=1}^{n_j} \ell_i^j$, où les ℓ_i^j sont des littéraux.

Exemple 2.55. La formule suivante est sous forme normale disjonctive :

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

La forme normale conjonctive est en général la plus adaptée lorsqu'on cherche un modèle d'une formule car il faut chercher une valuation satisfaisant chacune des clauses de la formule (le Lemme 2.56 ci-dessous précise ce point). Dans l'exemple 2.53, 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 2.56. *Soit φ une formule en FNC : $\varphi = \bigwedge_{i=1,\ldots,n} C_i$ où C_i sont des clauses. Pour tout $v \in \mathsf{Val}$, $v \in \mathsf{mod}(\varphi)$ ssi, pour tout $i = 1, \ldots, n$, il existe un littéral l in C_i tel que $v(l) = 1$.*

La preuve du Lemme étant immédiate, elle laissée au lecteur.

2.4.3.1 Mise sous forme clausale, en préservant l'équivalence

L'algorithme est le suivant :

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

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

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

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

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

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

Exemple 2.57. 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 :
$\varphi = (\neg(p \wedge (\neg q \vee (r \vee s)))) \wedge (p \vee q)$

— Etape 2 :
$\varphi = (\neg p \vee \neg(\neg q \vee (r \vee s))) \wedge (p \vee q)$
$\varphi = (\neg p \vee (\neg\neg q \wedge \neg(r \vee s))) \wedge (p \vee q)$
$\varphi = (\neg p \vee (q \wedge \neg(r \vee s))) \wedge (p \vee q)$
$\varphi = (\neg p \vee (q \wedge \neg r \wedge \neg s)) \wedge (p \vee q)$

— Etape 3 :
$\varphi = (\neg p \vee q) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee \neg s) \wedge (p \vee q)$

Proposition 2.58. *Le calcul précédent termine et donne une formule en forme clausale équivalente à la formule initiale.*

Remarque 2.59. Il n'y a pas unicité de la forme clausale.

2.4.3.2 Mise sous forme clausale, en préservant la satisfaisabilité

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

Proposition 2.60. *Soient $\varphi, \psi \in \mathcal{F}_{cp}$ et $p \notin \textsc{Prop}(\psi)$; supposons que φ ne contient pas des négations. On a alors que*

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

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

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

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

Exercice 2.61. Montrez que l'hypothèse que la négation n'apparaît pas dans φ est nécessaire.

Exercice 2.62. Montrez que les deux formules $\varphi[p \leftarrow \psi]$ et $\varphi \wedge (p \Rightarrow \psi)$ ne sont pas équivalentes.

Exemple 2.63. Considerez la formule

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

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

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

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

La Proposition 2.60 montre que la formule (2.1) est equisatisfaisable avec la formule

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

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

La Proposition 2.60 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 l'on est intéressé à l'existence d'un modèle seulement, non pas à énumérer tous les modèles. Parmi les défauts de la méthode, le fait qu'il introduit des nouveaux symboles propositionnels, ce qui alourdi évidemment les calculs de recherche d'un modèle.

2.5 Le problème SAT

2.5.1 Définition du problème

Définition 2.64 (Problème SAT)**.** Le problème SAT est le problème de décision qui consiste à déterminer si $\varphi \in \mathcal{F}_{\text{cp}}$ donnée en entrée admet, ou non, un modèle.

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

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

Exemple 2.65. Donnée : Une formule booléenne mise sous forme FNC :

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

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

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

2.5.2 Un problème NP-complet

Théorème 2.66. *Le problème SAT est décidable.*

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

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

- une recherche exhaustive comme dans l'algorithme précédent peut demander jusqu'à 2^n vérifications si φ possède n variables propositionnelles. Cette démarche est dite **déterministe**, mais son temps de calcul est exponentiel.
- d'un autre côté, si φ est satisfiable, il suffit d'une vérification à faire, à savoir tester précisément la configuration qui satisfait φ. Cette vérification demande un simple calcul booléen, qui se fait en temps polynomial (essentiellement linéaire en fait). Le temps de calcul cesse donc d'être exponentiel, à condition de savoir quelle configuration tester. Celle-ci pourrait par exemple être donnée par un être omniscient auquel on ne ferait pas totalement confiance. Une telle démarche est dite **non déterministe**.

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

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

2.5.3 Modélisation - Réduction à SAT

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

2.5.3.1 Comment modéliser

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

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

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

Exemple 2.67 (suite)**.**

Si et seulement si : Considérons l'énoncé

« Un triangle est équilatéral si, et seulement si, il a trois angles égaux. »

et posons :
- *equiangle* : le triangle a trois angles égaux
- *equilateral* : le triangle est équilatéral

L'enoncé se traduit par $equiangle \Leftrightarrow equilateral$. En effet, décomposons l'énoncé :
- « Un triangle est équilatéral si il a trois angles égaux » se traduit par $equiangle \Rightarrow equilateral$.
- « Un triangle est équilatéral seulement si il a trois angles égaux » signifie que si le triangle est équilatéral, alors il a forcément trois angles égaux et se traduit donc par $equilateral \Rightarrow equiangle$.

On a donc $equiangle \Rightarrow equilateral \wedge equilateral \Rightarrow equiangle$, ce qui est équivalent à $equilateral \Leftrightarrow equiangle$.

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

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

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

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

et considérons l'énoncé :

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

ou de façon équivalente :

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

Décomposons l'énoncé :

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

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

Donc $P_1 \equiv note \Rightarrow travail$.

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

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

Donc $P_2 \equiv travail \Rightarrow note$.

2.5.3.2 Formalisation du problème du Sudoku pour la réduction à SAT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.5.4 Algorithmes de résolution de SAT

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

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

Remarque 2.68. Nous pouvons identifier une formule en forme normale conjonctive φ avec l'ensemble $\mathcal{C}_\varphi$ des ses clauses, car évidemment nous avons $\mathsf{mod}(\varphi) = \mathsf{mod}(\mathcal{C}_\varphi)$, voir la Proposition 2.33. Par exemple, si

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

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

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

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

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

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

Remarque 2.69 (Simplifications)**.** Avant chercher un (ou plusieurs) modèle(s) d'une formule sous forme clausale (ou d'un ensemble de clauses), nous pouvons optimiser les calculs qui suivront en appliquant les règles suivantes :

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

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

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

Remarque 2.70 (Substitions simples)**.** Soit ψ une formule est sous forme clausale, et $\mathcal{C}$ l'ensemble de ses clauses. Le calcul d'une formule equivalente à $\psi[p \leftarrow \varphi]$, où $\varphi \in \{ \bot, \top \}$, peut se faire aisement via une procédure purement syntaxique sur l'ensemble $\mathcal{C}$. Définissons cette procedure :

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

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

Clairement, nous avons que

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

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

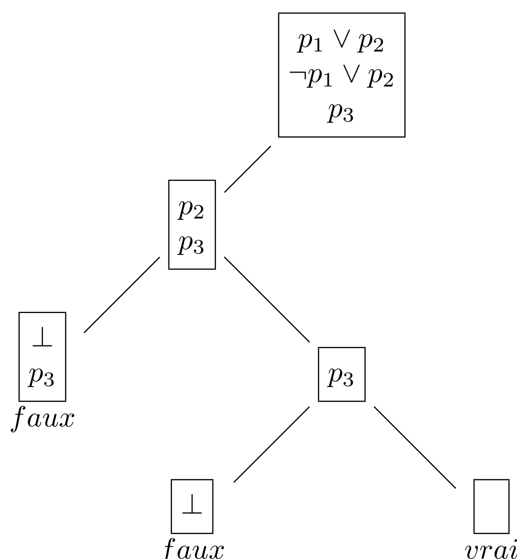
2.5.4.1 Algorithme de Quine

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

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

Algorithme de Quine
entrée : un ensemble de clauses $\mathcal{C}$ sortie : *vrai* si $\mathcal{C}$ est satisfaisable ou *faux* sinon
simplifier l'ensemble de clauses (cf. Remarque 2.69) ; si $\mathcal{C} = \varnothing$ retourner *vrai* si $\mathcal{C}$ contient la clause $\bot$ retourner *faux* choisir le prochain $p \in$ PROP apparaissant dans une clause si $Quine(\mathcal{C}[p \leftarrow \bot]) = vrai$ alors retourner *vrai* sinon retourner $Quine(\mathcal{C}[p \leftarrow \top])$

Exemple 2.71. Considerons $\{ p_1 \vee p_2, \neg p_1 \vee p_2, p_3 \}$. L'algorithme explore l'arbre de Herbrand selon un parcours en profondeur gauche comme suit :



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

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

2.5.4.2 Algorithme de Davis-Putnam-Logemann-Loveland

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

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

Algorithme $DPLL$	
entrée : un ensemble de clauses $\mathcal{C}$ sortie : $vrai$ si $\mathcal{C}$ est satisfaisable ou $faux$ sinon	
simplifier l'ensemble de clauses (cf. Remarque 2.69) ; si $\mathcal{C} = \varnothing$ retourner $vrai$ si $\mathcal{C}$ contient la clause $\bot$ retourner $faux$	
si $\mathcal{C}$ contient la clause unitaire p retourner $DPLL(\mathcal{C}[p \leftarrow \top])$ si $\mathcal{C}$ contient la clause unitaire $\neg p$ retourner $DPLL(\mathcal{C}[p \leftarrow \bot])$	(propagation des clauses unitaires)
choisir un littéral ℓ depuis une clause, **avec la bonne heuristique !**	(décision des littéraux)
si $DPLL(\mathcal{C}[\ell \leftarrow \bot]) = vrai$ alors retourner vrai	
sinon retourner $DPLL(\mathcal{C}[\ell \leftarrow \top])$	(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. Choisir les littéraux qui n'apparaissent que positivement et, parmi ceux-ci, choisir ceux qui sont présents le plus souvent ; les mettres à vrai.

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

Littéraux "courts". Choisir les littéraux apparaissant les plus, dans les clauses les plus courtes ; les mettre à faux.

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,

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

et, pour un symbole propositionnel p,

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

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

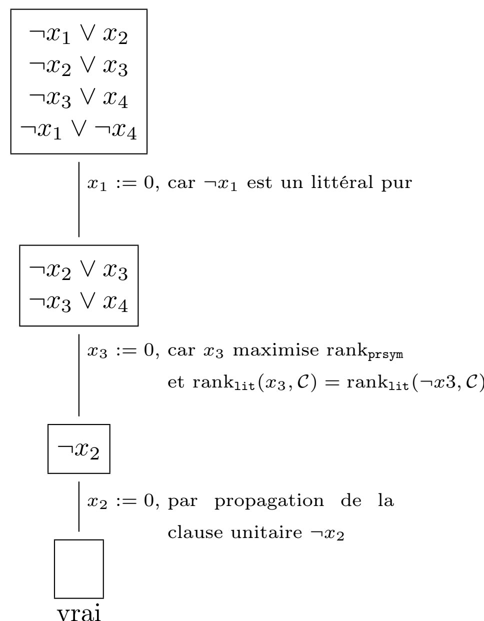
Notez que la définition que nous avons proposé n'est la seule. Par exemple, nous aurions pu définir

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

pour des pondérations α et β bien choisies.

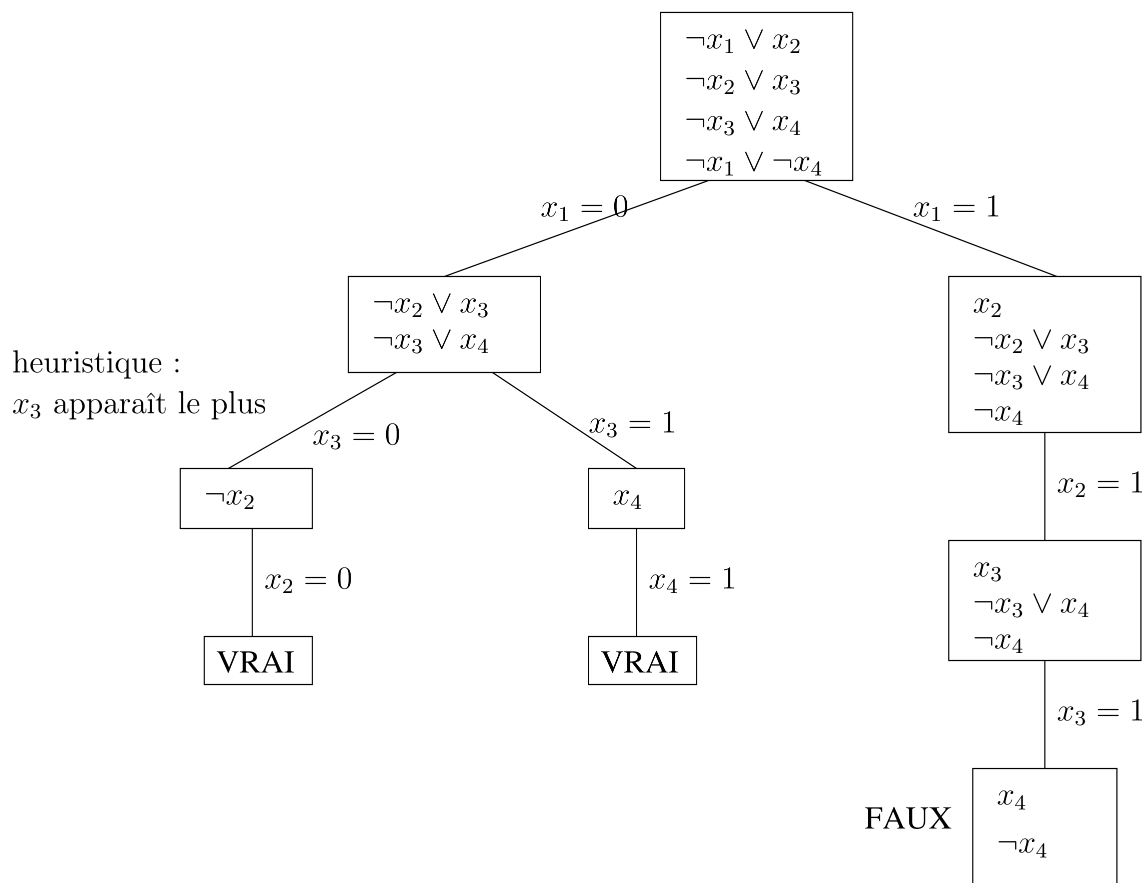
Remarque 2.72. Pour qu'une une heuristique soit effectivement implantée dans un SAT-solveur, il faut disposer de structures de données permettant de calculer le littéral choisi de façon efficace. Ainsi, il s'avère que les heuristique décrites ci-dessus ne fassent pas partie des solveurs existants les plus avancés (et performants).

Exemple 2.73. 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 2.74. Nous pouvons modifier l'algorithme DPLL (et bien sur, l'algorithme de Quine aussi), afin qu'il trouve tous les modèles d'un ensemble de clauses. Avec le même $\mathcal{C}$ de l'exemple précédent, l'algorithme produira le parcours suivant :



Les modèles de cette formule sont donc les suivants :

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

On remarque dans cet exemple que :

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

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

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

Exercice 2.75. Montrez que, pour tout $n \geq 1$, il existe un ensemble de clauses $\mathcal{C}_n$, dont les symboles propositionnels sont $\{ p_1, \ldots, p_n \}$ tel que l'arbre construit par l'algorithme de Quine est un arbre complet de profondeur n (donc, avec 2^n noeuds) où, par contre, l'arbre exploré par l'algorithme DPLL est une branche de longueur n (donc, seulement $n+1$ noeuds sont explorés par l'algorithme).

2.5.4.2.1 Lecture conseillées. L'implantation efficace de l'algorithme DPLL peut être assez subtile. Par exemple, l'heuristique des littéraux purs se révèle (à l'état actuel de connaissances) assez coûteuse ; pour cette raison, on préfère ne pas la mettre en ouvre. Le lecteur curieux pourra approfondir le sujet en lisant [ES04, GKSS07].

2.5.4.3 Algorithmes incomplets

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

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

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

2.5.4.4 Conclusion

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

2.5.5 Sous-classes de SAT

2.5.5.1 2-SAT

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

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

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

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

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

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

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

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

Une autre formulation est la suivante : la formule φ est satisfaisable si et seulement si pour chaque variable propositionnelle p, les sommets p et $\neg p$ du graphe 2-SAT sont dans deux composantes fortement connexes distinctes. (une composante fortement connexe d'un graphe orienté G est un sous-graphe maximal de G tel que pour toute paire de sommets u et v dans ce sous-graphe, il existe un chemin de u à v et un chemin de v à u.)

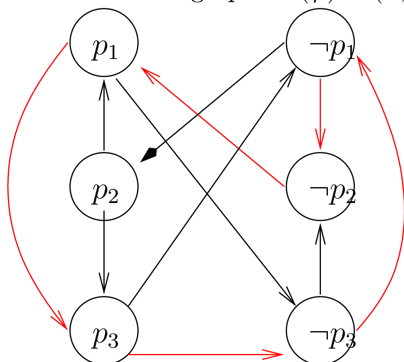
L'algorithme de Tarjan permet de calculer les composantes fortement connexes d'un graphe orienté en $\mathcal{O}(|S| + |A|)$, donc 2-SAT est bien polynomial.

Exemple 2.76. $\varphi = (p_1 \vee p_2) \wedge (\neg p_1 \vee p_3) \wedge (\neg p_2 \vee p_1) \wedge (\neg p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_3)$

On a :

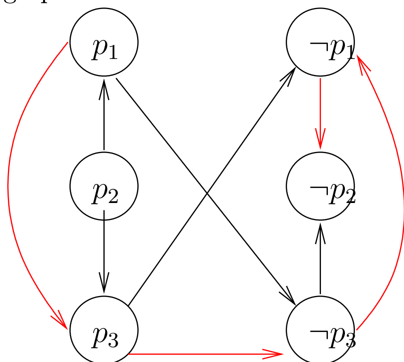
- $(p_1 \vee p_2) \equiv (\neg p_1 \Rightarrow p_2) \wedge (\neg p_2 \Rightarrow p_1)$
- $(\neg p_1 \vee p_3) \equiv (p_1 \Rightarrow p_3) \wedge (\neg p_3 \Rightarrow \neg p_1)$
- $(\neg p_2 \vee p_1) \equiv (p_2 \Rightarrow p_1) \wedge (\neg p_1 \Rightarrow \neg p_2)$
- $(\neg p_2 \vee p_3) \equiv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

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



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

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



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

2.5.5.2 3-SAT

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

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

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

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

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

2.5.5.3 Horn-SAT

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

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

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

(c) p ;

(d) $\perp$;

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 :
— $\neg p_1 \vee \ldots \vee \neg p_n \vee p \equiv (p_1 \wedge p_2 \wedge \ldots \wedge p_n) \Rightarrow p$ (avec $n > 0$) ;
— $\neg p_1 \vee \ldots \vee \neg p_n \equiv (p_1 \wedge p_2 \wedge \ldots \wedge p_n) \Rightarrow \perp$ avec $n > 0$;

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

Les clauses réduites à une variable propositionnelle sont appelés faits.

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

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

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

2.5.6 Les SAT-solvers

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

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

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

2.5.7 Applications

2.5.7.1 Vérification d'un circuit logique

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

2.5.7.2 Satisfaction de contraintes

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

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

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

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

2.5.7.3 Diagnostic

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

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

On peut simplifier la modélisation par des formules comme les suivantes :
$\neg Ab(S) \Rightarrow Int1 \wedge Obs1$
$Ab(S) \Rightarrow Int2 \wedge Obs2$

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

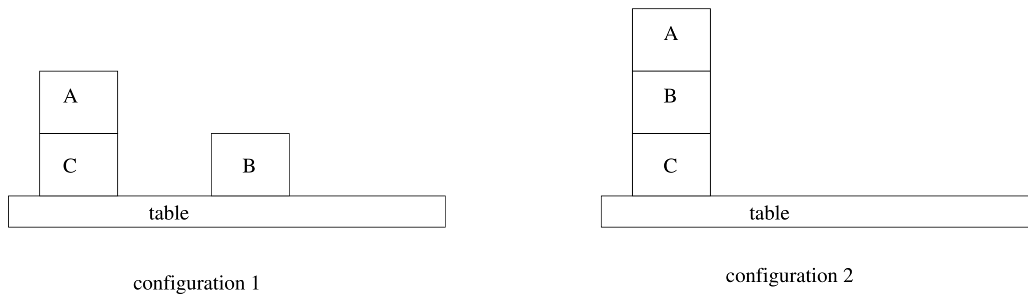
2.5.7.4 Planification

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

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

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

Chaque action spécifie généralement des préconditions qui doivent être présentes dans l'état actuel pour qu'elle puisse être appliquée, et des postconditions (effets sur l'état actuel). Le problème de planification classique consiste à trouver une séquence d'actions menant d'un état du système à un ensemble d'états. Par exemple, voici un problème de planification simple. On dispose de boites sur une tables, dans une configuration initiale, et on souhaite obtenir une nouvelle configuration (par exemple passer de conf 1 de la figure à a conf 2). La seule action possible est de déplacer une boite non recouverte par une autre sur la table ou sur une autre boite.

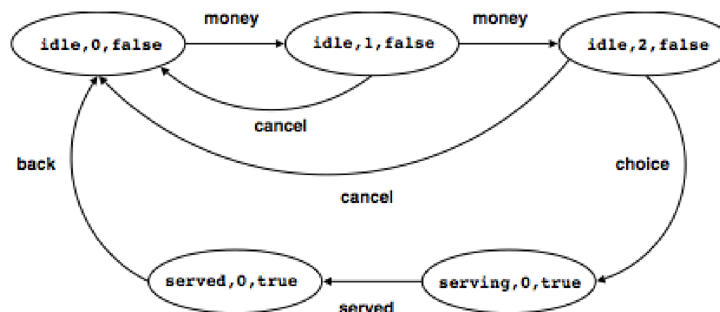


2.5.7.5 Model checking

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

Représentation formelle : $\mathcal{M} \models^? \varphi$
— $\mathcal{M}$: (modèle du) programme sous observation
— φ : propriété à vérifier
— pré-requis : sémantique (opérationnelle), langage de spécification

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



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

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

2.5.7.6 Cryptographie

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

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

2.5.7.7 Bio-informatique

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

2.5.8 Sur la modélisation

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

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

2.6 Systèmes de preuves

2.6.1 La notion de système formel

Un *système de preuves* définit des règles 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 règles d'inférence permet de déduire une nouvelle formule, la conclusion, à partir d'un ensemble de formules, les prémisses ou hypothèses.

On note $\vdash \varphi$ si φ peut se déduire grace au système de preuves, i.e., 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** : une formule déduite est une tautologie ?
2. **complet** : toute tautologie peut-elle se déduire ?

Par extension, un *système formel* est un ensemble de règles permettant d'inférer 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 2.7) 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 montreront qu'on peut déduire le couple (Γ, Δ) dans le calcul si est seulement si la formule $\bigvee_{\delta \in \Delta} \delta$ est une conséquence logique de Γ.

2.6.2 La méthode de la coupure

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

2.6.2.1 Le système

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

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

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

$$\frac{C \vee \ell \vee \ell}{C \vee \ell}\ \text{factorisation}$$

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

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

On peut utiliser ce système formel pour :

1. montrer qu'une formule (ou un ensemble de formules) est contradictoire (ou admet un modelé) ;
2. montrer qu'une formule est une tautologie (ou non) : on montre que sa négation est contradictoire. On dit que la méthode de la coupure est *réfutationnelle* : pour prouver la formuel φ, on suppose $\neg\varphi$ et on montre que cela conduit à une contradiction ;
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 2.77. Soit la formule $\varphi = (p \lor r \lor s) \land (r \lor \neg s) \land \neg r \land \neg p$; cette formule s'écrit comme l'ensemble de clauses $\mathcal{C} = \{(p \lor r \lor s),\ (r \lor \neg s),\ \neg r,\ \neg p\}$. Voici une dérivation par résolution à partir de cet ensemble de clauses :

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

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

Exemple 2.78. L'exemple suivant montre qu'on peut avoir besoin d'utiliser deux fois une clause pour dériver la clause vide. Soit φ la formule $((p \Rightarrow q) \land (q \Rightarrow p) \land (p \lor q)) \Rightarrow (p \land q)$. On veut montrer que φ est est une tautologie, on met donc $\neg\varphi$ sous forme clausale, on obtient l'ensemble de clause $\mathcal{C} = \{(p \lor \neg q), (\neg p \lor q), (p \lor q), (\neg p \lor \neg q)\}$. Une preuve par la méthode de la coupure est la suivante :

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

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

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

Remarque 2.79. 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 :

$$\cfrac{p \lor \neg q \qquad \neg q \lor \neg p}{\neg q \lor \neg q}\ \text{coupure}$$

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

2.6.2.2 Correction et complétude

Notation 2.80. 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'ensemble de clauses $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$ 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 $\psi_n = \psi$.

Théorème 2.81. *La résolution est correcte ; c'est-à-dire, si $\mathcal{C} \vdash_R \bot$, alors $\mathcal{C}$ est insatisfaisable.*

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

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

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

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

Définition 2.82. Un ensemble de clauses (factorisées) est dit *saturé* si on ne peut pas produire des nouvelles clauses par application de la règle de coupure.

Théorème 2.83. *La résolution est complète. C'est-à-dire : si $\mathcal{C}$ est insatisfaisable, alors $\mathcal{C} \vdash_R \bot$.*

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

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

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

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

On suppose donc que $\mathcal{S}$ est saturé et ne contient pas la clause $\bot$ et on montre que $\mathcal{S}$ a un modèle. On fait la preuve par récurrence sur le nombre n de symboles propositionnels de $\mathcal{S}$.

Cas de base : $n = 1$. 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 donc la clause vide. Donc $\mathcal{S}$ admet un modèle.

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

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

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

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

comme suggéré par la Figure 2.6.2.2. De même, $\mathcal{S}'_1$ est saturé, et $\bot \in \mathcal{S}'_1$ implique $\neg p \in \mathcal{S}_1$. En particulier, si $\bot \in \mathcal{S}'_0 \cap \mathcal{S}'_1$, alors $p \in \mathcal{S}_0 \subseteq \mathcal{S}$ et $\neg p \in \mathcal{S}_1 \subseteq \mathcal{S}$; car $\mathcal{S}$ est saturé, alors on obtient $\bot \in \mathcal{S}$, une contradiction.

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

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

FIGURE 2.5 – Commutation de la coupure par rapport à la subsitution

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 $1 = u(p) \leq u(C)$. □

2.6.2.3 L'algorithme de résolution

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

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

L'algorithme termine lorsque :

(a) soit on vient d'ajouter la clause $\bot$ 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 $\bot$ — peut se considerer 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 résolvantes qui ne sont pas déjà dans l'ensemble de clauses, soit avoir un critère permettant d'arrêter de générer de nouvelles résolvantes.

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 2.84. 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 2.85. $\mathcal{C} = \{p \vee \neg p \vee \neg q\}$. 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 $p \vee \neg p \vee \neg q \vee \neg q$.

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

La méthode va générer les clauses de la forme $p \vee \neg p \vee \neg q \vee \neg q \vee \ldots \neg q$ et aucune autre (la encore un raisonnement est à faire). Donc la méthode ne termine pas et engendre un ensemble de

clauses infini qui ne contient pas la clause vide. On peut conclure soit en trouvant directement un modèle (on peut en trouver plusieurs ici) soit en appliquant 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 2.86. Un clause C est *factorisée* si elle ne contient pas deux occurrences du même littéral.

Si C est une clause, nous allons dénoter par $\mathfrak{f}(C)$ la clause factorisée obtenue de C par une suite de règles de factorisation. L'algorithme se présente donc comme en Figure 2.6. La terminaison de

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

FIGURE 2.6 – L'algorithme de coupure (ou de résolution)

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.

Exercice 2.87. Estimer cette borne supérieure.

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

2.6.3 Systèmes de preuve à la Hilbert

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

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

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

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

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

Nous allons donc présenter un deuxième système formel pour la logique propositionnelle, qui étudie et simule le raisonnement mathématique de près, en donnant à l'implication son rôle traditionnel de premier plan. Rappelez vous que toute formule est équivalent à (donc peut se coder

1. La structure de l'inférence exemplifiée est en fait bien plus complexe ; cet exemple peux pleinement se comprendre dans le cadre de la logique du premier ordre.

dans) une formule construite à partir de l'implication et de la négation. On peut donc supposer que les connecteurs logique sont ces deux seulement.

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

Axiomes. Sont les suivants :

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

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

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

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

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

Définition 2.88. Soit $\Gamma \subseteq \mathcal{F}_{\text{cp}}$ et $\varphi \in \mathcal{F}_{\text{cp}}$. Une démonstration (ou preuve) de φ à partir des hypothèses Γ est une liste ordonnée $\varphi_1, \ldots, \varphi_n$ telle que

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

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

Exemple 2.89. Soit $\varphi, \psi \in \mathcal{F}_{\text{cp}}$. Nous avons $\vdash \varphi \Rightarrow (\psi \Rightarrow \varphi)$ car la liste de longuer 1 composée par $\varphi \Rightarrow (\psi \Rightarrow \varphi)$ est une preuve de cette formule.

Exemple 2.90. Considérons la suite des formules suivantes :

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

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

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

Exemple 2.91. Clairement, nous avons $\{ X, X \Rightarrow Y \} \vdash Y$.

Exemple 2.92. Nous avons $\{ X \Rightarrow Y, Y \Rightarrow Z \} \vdash X \Rightarrow Z$, pour tout $X, Y, Z \in \mathcal{F}_{\text{cp}}$. Voici la (quasi)-démonstration qui en est témoin :

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

2. On devrait donc parler de *schémas d'axiomes* au lieu d'*axiome*.

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

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

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

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

Lemme 2.93. *Soit* $\Pi = \varphi_1, \ldots, \varphi_n$ *une suite de formules telles que, pour tout* $i = 1, \ldots, n$, *on a un de ces cas :*

— φ_i *est un axiome,*
— $\varphi_i \in \Gamma$,
— φ_i *est inférée à partir des formules* φ_i, φ_k, *avec* $j, k < i$, *via la règle MP,*
— $\Gamma \vdash \varphi_i$,
— $\Gamma \cup \Sigma \vdash \varphi_i$, *où* Σ *est un sous-ensemble de* $\{ \varphi_1, \ldots, \varphi_{i-1} \}$.

Alors $\Gamma \vdash \varphi_n$.

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

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

Exemple 2.94. Comme exemple du Lemme 2.93, nous allons montrer que la lois d'involution

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

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

1,	$\neg\psi \Rightarrow \neg\psi$	car $\vdash \neg\psi \Rightarrow \neg\psi$, voir 2.90
2,	$(\neg\psi \Rightarrow \neg\psi) \Rightarrow ((\neg\psi \Rightarrow \neg\neg\psi) \Rightarrow \psi)$	A3, avec ψ et $\neg\psi$
3,	$(\neg\psi \Rightarrow \neg\neg\psi) \Rightarrow \psi$	MP, de 1 et 2
4,	$\neg\neg\psi \Rightarrow (\neg\psi \Rightarrow \neg\neg\psi)$	A1, avec $\neg\neg\psi$ et $\neg\psi$
5,	$\neg\neg\psi \Rightarrow \psi$	MB, de 4 et 3

Le théorème de Déduction

Théorème 2.95 (de Déduction)**.** *On a*

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

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

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

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

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

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

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

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

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

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

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

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

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

TODO : ajouter discussion de ce théorème

Le théorème de Correction et Complétude

Lemme 2.96. *Les axiomes A1, A2, et A3, sont des tautologies, pour tout instanciation de X, Y, Z par des formules.*

Proposition 2.97 (Correction)**.** *Si $\Gamma \vdash \varphi$, alors $\Gamma \models \varphi$.*

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

Pour $i = 1$, alors $\varphi_i \in \Gamma$, ou bien φ_i est un axiome. Si $\varphi_i \in \Gamma$, alors $v(\varphi_i) = 1$ car $v \in \mathsf{mod}(\Gamma)$, et si φ_i est un axiome, alors $v(\varphi_i) = 1$ par le Lemme 2.96.

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

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

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

Lemme 2.98. *Pour toutes formules $\varphi, \psi \in \mathcal{F}_{cp}$, nous avons :*

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

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

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

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

Proposition 2.99 (Complétude)**.** *Si* $\Gamma \models \varphi$*, alors* $\Gamma \vdash \varphi$*.*

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

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

$$\Gamma_{n+1} = \begin{cases} \Gamma_n \cup \{\,\varphi_{n+1}\,\}\,, & \text{si } \Gamma_n \cup \{\,\varphi_{n+1}\,\} \not\vdash \varphi\,, \\ \Gamma_n\,, & \text{sinon}\,. \end{cases}$$

Posons enfin

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

Remarquons que $\Gamma_n \not\vdash \varphi$, pour tout $n \geq 0$, et aussi les propriétés suivantes de Γ_ω :

1. $\Gamma_\omega \not\vdash \varphi$. Car sinon, il existe un sous-ensemble fini $\Gamma' \subseteq \Gamma_\omega$ tel que $\Gamma' \vdash \varphi$, et ainsi il existe $n \geq 0$ avec $\Gamma' \subseteq \Gamma_n$; cela entraîne que $\Gamma_n \vdash \varphi$, ce qui contredit la définition de Γ_n.
2. $\Gamma_\omega \cup \{\,\psi\,\} \vdash \varphi$, pour toute formule $\psi \notin \Gamma_\omega$. En fait supposons, que $\psi \notin \Gamma_\omega$ et $\psi = \varphi_{n+1}$. Or car $\psi \notin \Gamma_\omega$, cela implique que $\Gamma_n = \Gamma_{n+1}$, et la seule raison de cette identité est que $\Gamma_n \cup \{\,\psi\,\} \vdash \varphi$. A fortiori, $\Gamma_\omega \cup \{\,\psi\,\} \vdash \varphi$.
3. $\psi \in \Gamma_\omega$ ssi $\neg\psi \notin \Gamma_\omega$. En fait, si $\psi, \neg\psi \in \Gamma_\omega$, alors $\Gamma_\omega \vdash \varphi$, ce qui contredit (1) ; si par contre $\psi, \neg\psi \notin \Gamma_\omega$, alors $\Gamma_\omega \cup \{\,\psi_n\,\} \vdash \varphi$ et $\Gamma_\omega \cup \{\,\neg\psi_n\,\} \vdash \varphi$; par le Lemme 2.98, nous avons encore $\Gamma_\omega \vdash \varphi$.
4. $\Gamma_\omega \vdash \psi$ implique $\psi \in \Gamma_\omega$. Si $\psi \notin \Gamma_\omega$, alors $\Gamma_\omega \cup \{\,\psi\,\} \vdash \varphi$. Par le Lemme 2.98, on a alors $\Gamma_\omega \vdash \varphi$.

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

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

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

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

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

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

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

Nous pouvons finalement énoncer le Théorème de Correction et Complétude :

Théorème 2.100. *La relation* $\Gamma \vdash \varphi$ *est vraie si, et seulement si, la relation* $\Gamma \models \varphi$ *est vraie.*

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

TODO : Ajouter discussion de ces Théorèmes

2.7 Les règles du calcul des séquents

Règles structurelles

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

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

Règles logiques

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

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

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

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

Règle de coupure

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

2.8 Résumé

Formules propositionnelles

Les briques de base des formules propositionnelles sont les **propositions** appelées aussi **symboles propositionnels** ou **atomes** ou **formules atomiques** ou **variables propositionnelles**. On note Prop l'ensemble des propositions. L'ensemble Prop_0 des **formules propositionnelles** est le plus petit ensemble contenant Prop et clos par l'application des connecteurs $\wedge, \vee, \neg, \Rightarrow$.

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

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

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

où les ℓ_i et ℓ_i^j sont des littéraux.

Modèles d'une formule

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

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

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

Modèles d'un ensemble de formules

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

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

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

Système de preuve.

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

Bibliographie

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

[GKSS07] C.P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. *Handbook of Knowledge Representation*, chapter Satisfiability solvers. Elsevier, 2007.

Chapitre 3

Calcul des prédicats

3.1 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 lieux, il contient des individus (ou entités) (donnés par des symboles de variables $x, y, z, \ldots$). Il contient des fonctions $(f, g, \ldots, s, \ldots)$ permettant de transformer des entités en autres entités (par exemple la fonction qui associe une personne à son père), et des relations $(\ldots, P, Q, R, \ldots)$ 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 les interprétations comportent en général une infinité d'individus, il est alors difficile de vérifier que $\forall x \varphi$ puisque x peut prendre une infinité de valeurs différentes.

3.2 Préliminaires

On rappelle ici les notions de bases sur les fonctions et relations.

3.2.1 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 une 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 3.1.

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 $f(n) = n + 1$.

Une fonction d'arité 0 sur E est une constante $c \in E$.

3.2.2 Les relations

Etant donné un ensemble E, et n un entier positif, une relation n-aire (ou d'arité n) sur E est un sous-ensemble de E^n.

Exemple 3.2.

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 $R = \{1, 2\}$.

Une relation d'arité 0 sur E est un ensemble vide puisque c'est un sous-ensemble de E^0.
Si R est une relation d'arité n, on note $R(a_1, \ldots, a_n)$ ssi $(a_1, \ldots, a_n) \in R$.

3.3 Un exemple

Avant d'en venir aux définitions formelles, considérons les formules suivantes :
$\varphi_G : \forall x \forall y \forall z (P(x, y) \wedge P(y, z)) \Rightarrow G(x, z)$
$\varphi_P : \forall x \exists y P(y, x)$
$\varphi_C : \forall x \exists y G(y, x)$
$\varphi_D : \forall x \forall z P(z, f(x)) \Rightarrow G(z, x)$
$\varphi_F : (\varphi_G \wedge \varphi_P) \Rightarrow \varphi_C$.
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 connait pas la valeur de la fonction $f : D \to D$
— on ne connait 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.

3.3.1 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).

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, ...

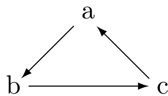
3.3.2 Interprétation 2

Les individus sont trois points a, b, c. On interprète les prédicats par les relations suivantes :

La relation P est vraie pour les couples (a, b), (b, c) et (c, a) et fausse pour les autres. En d'autres termes, $P(a, b)$, $P(b, c)$, $P(c, a)$ sont vraies, et $P(a, a)$, $P(a, c)$, $P(a, b)$, $P(b, b)$, $P(c, b)$, $P(c, c)$ sont fausses.

La relation G est vraie pour les couples (b, a), (c, b) et (a, c) et fausse pour les autres.

La fonction f est définie par $f(a) = a$, $f(b) = b$ et $f(c) = a$.



En représentant a, b et c par les points suivants, $P(x, y)$ signifie « x précède immédiatement y » et $G(x, y)$ signifie « x suit immédiatement y ».

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 également vraie.

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.

La formule φ_D signifie que pour tout z et pour tout x, si z précède immédiatement $f(x)$, alors z suit immédiatement x. Elle est donc fausse : c précède $f(a) = a$ sans que c suive immédiatement a.

3.3.3 Interprétation 3

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 fausse. 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.

3.3.4 Interprétation 4

Les individus sont encore les quatre sommets a, b, c, d du graphe précédent.

$P(x, y)$ signifie que x précède immédiatement y. $G(x, y)$ signifie que pour aller de x à y on rencontre exactement un point z différent de x et de y.

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 vraie.

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.

3.3.5 Interprétation 5

Les individus sont les nombres entiers positifs.

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

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

La formule φ_P signifie alors que pour tout entier y, il existe un entier x tel que $x = y + 1$. Elle est vraie.

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

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

3.3.6 Interprétation 6

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 fausse.

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

La formule φ_P signifie alors que pour tout entier y, il existe un entier x tel que $y = x + 1$. Elle est fausse : 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 fausse : 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.

3.3.7 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.

3.4 Expressions et formules

3.4.1 Les termes

Définition 3.3 (Signature)**.** Une *signature* est un ensemble $\mathcal{S}$ de symboles muni d'une application $\rho : \mathcal{S} \to \mathbb{N}$, appelée *arité*.

On écrira une signature comme un ensemble liste 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 appellé *constante*.

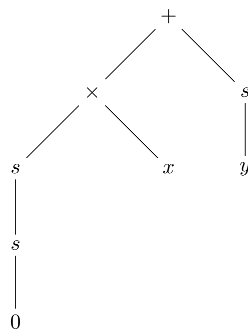
Définition 3.4 (Termes)**.** Étant donné une signature $\mathcal{S}$ et un ensemble X (de variables individuelles), l'ensemble $\mathcal{T}_{\mathcal{S}}(X)$ des termes est le plus petit ensemble tel que :

— $x \in \mathcal{T}_{\mathcal{S}}(X)$ pour toute variable $x \in X$

— si $f \in \mathcal{S}$ est d'arité $n \geq 0$ et si $t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{S}}(X)$, alors $f(t_1, \ldots, t_n) \in \mathcal{T}_{\mathcal{S}}(X)$.

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 a 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 3.5. 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,

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

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

Exemple 3.6. 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}.

3.4.2 Le langage

Définition 3.7 (Langage)**.** Un *langage* (ou *vocabulaire*) du premier ordre est la donnée d'un couple de signatures $\mathcal{S} = (\mathcal{S}_{\mathtt{f}}, \mathcal{S}_{\mathtt{r}})$ disjointes (i.e. avec $\mathcal{S}_{\mathtt{f}} \cap \mathcal{S}_{\mathtt{r}} = \varnothing$).

On dit que :
— Les elements de $\mathcal{S}_{\mathtt{f}}$ sont les *symboles de fonction* du langage.
— Les elements de $\mathcal{S}_{\mathtt{r}}$ sont les *symboles de relation* (ou de prédicat) du langage.

Remarque 3.8.

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 3.9.

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 : $\varnothing$;
   — de fonction : $\cap$ et $\cup$ binaires, C unaire (le complément) ;
   — de relation : $=$, $\in$ et $\subset$, tous binaires. [2]

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

3.4.3 Les formules du calcul des prédicats

Etant donné un langage $\mathcal{S} = (\mathcal{S}_\mathtt{f}, \mathcal{S}_\mathtt{r})$, 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 $X = \{x, x_0, x_1, \ldots, y, y_0, y_1, \ldots, z, z_0, z_1, \ldots\}$ de variables.

Définition 3.10 (Formules atomiques)**.** Soit $\mathcal{S} = (\mathcal{S}_\mathtt{f}, \mathcal{S}_\mathtt{r})$ un langage, une formule atomique sur $\mathcal{S}$ est une expression de la forme suivante :

— $r(t_1, \ldots, t_n)$ où $r \in \mathcal{S}_\mathtt{r}$ est d'arité $n \geq 0$ et $t_1, t_2, \ldots, t_n \in \mathcal{T}_{\mathcal{S}_\mathtt{f}}(X)$ sont des termes.

Définition 3.11 (Formules)**.** Etant donné un langage $\mathcal{S}$, l'ensemble $\mathcal{F}_{\text{po}}(\mathcal{S})$ des formules du premier ordre sur $\mathcal{S}$ est le plus petit ensemble tel que :

1. toute formule atomique sur $\mathcal{S}$ appartient à $\mathcal{F}_{\text{po}}(\mathcal{S})$,
2. si φ, $\psi \in \mathcal{F}_{\text{po}}(\mathcal{S})$ sont deux formules alors :
   — $\varphi \wedge \psi \in \mathcal{F}_{\text{po}}(\mathcal{S})$,
   — $\varphi \vee \psi \in \mathcal{F}_{\text{po}}(\mathcal{S})$,
   — $\varphi \Rightarrow \psi \in \mathcal{F}_{\text{po}}(\mathcal{S})$,
   — $\neg\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$,
3. si $\varphi \in \mathcal{F}_{\text{po}}$ et $x \in X$ alors
   — $\forall x\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$,
   — $\exists x\varphi \in \mathcal{F}_{\text{po}}(\mathcal{S})$.

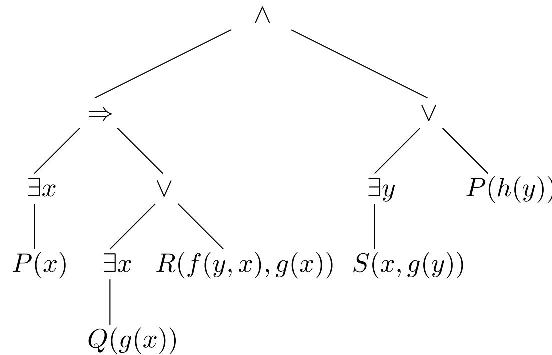
Exemple 3.12. Reprenons les langages de l'exemple 3.9 :

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 $\mathcal{L}_2$, $\forall x \forall y(x \cap y = \varnothing \Rightarrow (x = \varnothing \wedge y = \varnothing))$ 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

$$[(\exists x P(x)) \Rightarrow (R(f(y, x), g(x)) \vee [\exists x Q(g(x))])] \wedge [(\exists y S(x, g(y))) \vee P(h(y))] \tag{3.1}$$

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



3.4.4 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 $\mathsf{FV}(\varphi)$ (depuis l'anglais, *free variable*), et l'ensemble des variables liées $\mathsf{BV}(\varphi)$ (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 tout occurrence d'une variable x dans φ est libre : $\mathsf{FV}(\varphi) = Var(\varphi)$ et $\mathsf{BV}(\varphi) = \varnothing$.

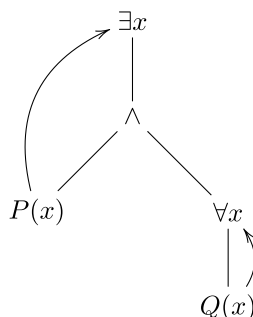
— si φ est $\exists x\psi$ ou $\forall x\psi$, alors $\mathsf{FV}(\varphi) = \mathsf{FV}(\psi) - \{x\}$; $\mathsf{BV}(\varphi) = \mathsf{BV}(\varphi) \cup \{x\}$ (et toute occurrence de x **libre dans** ψ devient liée dans φ par le quantificateur introduit) ;

— si $\varphi = \varphi_1 \circ \varphi_2$ (où $\circ \in \{\wedge, \vee, \Rightarrow\}$) alors $\mathsf{FV}(\varphi) = \mathsf{FV}(\varphi_1) \cup \mathsf{FV}(\varphi_2)$, $\mathsf{BV}(\varphi) = \mathsf{BV}(\varphi_1) \cup \mathsf{BV}(\varphi_2)$;

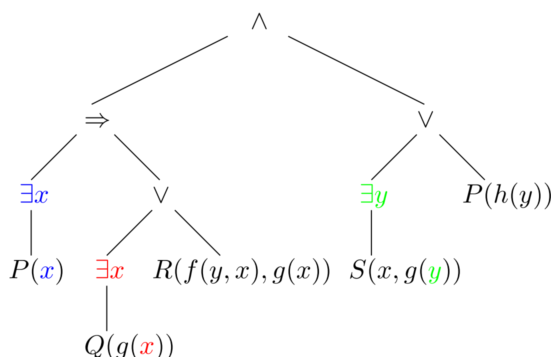
— si $\varphi = \neg\psi$, alors $\mathsf{FV}(\varphi) = \mathsf{FV}(\psi)$ et $\mathsf{BV}(\varphi) = \mathsf{BV}(\psi)$.

Définition 3.13 (Formule close)**.** Une formule φ est dite **close** ssi $\mathsf{FV}(\varphi) = \varnothing$.

Exemple 3.14. Une variable liée est attachée à une et une seule occurence d'un quantificateur dans la formule : celui qui dans l'arbre est son ancêtre le plus proche. Par exemple, considérons la formule $\exists x P(x) \wedge \forall x Q(x)$. Les variables sont liées de la façon suivante :

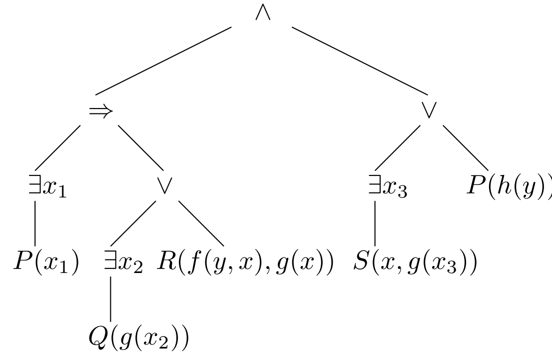


Exemple 3.15. Considérons la formule (3.1) et son arbre :



3. Nous ne pouvons pas utiliser un acronyme français, car on obtiendrait le même acronyme pour les variables libres et celles liées.

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 precedent, nous observons qu'il peut bien etre le cas que $\mathsf{FV}(\varphi) \cap \mathsf{BV}(\varphi) \neq \varnothing$. 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 ambiguités. Ainsi, nous pourrons toujours supposer que $\mathsf{FV}(\varphi) \cap \mathsf{BV}(\varphi) = \varnothing$. Par exemple, la formule précédente peut-être renommée de la façon suivante :



La nouvelle formule est donc

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

3.5 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.

3.5.1 Structures

Notation 3.16. 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 :

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

Remarque 3.17. Remarquez que nous pouvons donner un sens à l'ensemble D^n avec $n = 0$:

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

En particulier, le lecteur observera que les fonctions $f : D^0 \to D$ sont en bijection avec les éléments $d \in D$ (via l'évaluation $f() = d \in D$). Nous allons donc identifier fonctions $f : D^0 \to D$ (elles sont les fonctions constantes, qui ne dépendent pas de paramètres) et éléments $d \in D$.

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 3.18. Soit $\mathcal{S} = (\mathcal{S}_{\mathtt{r}}, \mathcal{S}_{\mathtt{f}})$ 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 $R \in \mathcal{S}_{\mathtt{r}}$;

(b) d'une fonction totale (application) $f^{\mathcal{M}} : D_{\mathcal{M}}^{\rho(f)} \to D_{\mathcal{M}}$, pour chaque symbole de fonction $f \in \mathcal{S}_{\mathtt{f}}$.

Remarque 3.19. A cause de la remarque 3.17, 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) $c \in \mathcal{S}_{\mathtt{f}}$,

(b.2) d'une fonction totale (application) $f^{\mathcal{M}} : D_{\mathcal{M}}^{\rho(f)} \to D_{\mathcal{M}}$, pour chaque symbole de fonction $f \in \mathcal{S}_{\mathtt{f}}$, tel que $\rho(f) \geq 1$.

Exemple 3.20. 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 3.21. Supposons que $\mathcal{S}_{\mathtt{f}} = \{ (o, 0), (s, 1) \}$ et $\mathcal{S}_{\mathtt{r}} = \{ (\mathtt{Even}, 1) \}$. 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

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

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.*

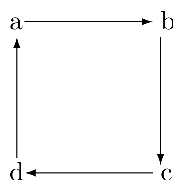
La situation est assez différente quand le symbole d'égalité est parmi les symboles de relation. Dans ce cas, on interprétera, *par défaut*, la relation d'égalité sur l'image de la fonction diagonale :

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

Exemple 3.22. 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'aupravant, mais maintenant B est interprété comme la relation "ne pas être voisin direct" : $B^{\mathcal{M}} = \{(b, d), (d, b), (a, c), (c, a)\}$. Ce modèle, même s'il partage avec le précédent le domaine, est différent du précédent.
3. Le domaine est l'ensemble des triangles et B est la relation "avoir la même aire"

Exemple 3.23. Supposons que $\rho(R) \leq 2$ pour tout $R \in \mathcal{S}_{\mathtt{r}}$, et que $\rho(f) \leq 1$ pour tout $f \in \mathcal{S}_{\mathtt{f}}$. 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 $P \in \mathcal{S}_{\mathtt{r}}$ si $d \in P^{\mathcal{M}}$;
— un noeud $d \in D_{\mathcal{M}}$ est étiqueté par un symbole de constante $c \in \mathcal{S}_{\mathtt{f}}$ 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 $R \in \mathcal{S}_{\mathtt{r}}$; une arête $d \to d'$ est étiqueté par les symboles $R \in \mathcal{S}_{\mathtt{r}}$ tels quel $(d, d') \in R$;
— on met une arête pointillée de d vers d' si $f(d) = d'$; une arête pointillée $d \to d'$ est étiqueté par les symboles $f \in \mathcal{S}_{\mathtt{f}}$ tels quel $f(d) = d'$;

Par exemple, si $\mathcal{S}_{\mathtt{r}} = \{ (P, 2) \}$ et $\mathcal{S}_{\mathtt{f}} = \{ (f, 1) \}$, 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 3.1.

Exemple 3.24. 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
$\{0\}$	$\varnothing$	$\{p_0, \ldots, p_n, \ldots\}$	Structure propositionnelle
$\mathcal{T}_{\mathcal{S}_\mathtt{f}}(\varnothing)$	$\hat{\mathcal{S}}_\mathtt{f}$	$\hat{\mathcal{S}}_\mathtt{r}$	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 $\hat{f} \in \hat{\mathcal{S}}_\mathtt{f}$ d'arité n associe aux termes $t_1, \ldots, t_n$ le nouveau terme $f(t_1, \ldots, t_n)$, et les relations sont quelconques.

3.5.2 Evaluation (des termes et) des formules

Comme pour le calcul des propositions, on va 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 3.25. 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 $\mathcal{V} : X \to D_{\mathcal{M}}$.

On définit maintenant la valeur $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}}$ 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 3.26. Soit $\mathcal{M}$ une $\mathcal{S}$-structure et $\mathcal{V}$ une valuation de X dans $D_{\mathcal{M}}$; la valeur d'un terme $t \in \mathcal{T}_{\mathcal{S}_\mathtt{f}}(X)$, notée $[\![t]\!]_{\mathcal{M},\mathcal{V}}$ est un élément de $D_{\mathcal{M}}$ défini (par induction) par :
— pour toute variable $x \in X$, $[\![x]\!]_{\mathcal{M},\mathcal{V}} = \mathcal{V}(x)$;
— pour tout $f \in \mathcal{S}_\mathtt{f}$ d'arité $n \geq 0$, pour tous termes $t_1, \ldots t_n$,

$$[\![f(t_1, \ldots, t_n)]\!]_{\mathcal{M},\mathcal{V}} := f^{\mathcal{M}}([\![t_1]\!]_{\mathcal{M},\mathcal{V}}, \ldots, [\![t_n]\!]_{\mathcal{M},\mathcal{V}}) .$$

Remarque 3.27. 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 $c \in \mathcal{S}_\mathtt{f}$ (c'est-à-dire symbole de fonction d'arité 0), $[\![c()]\!]_{\mathcal{M},\mathcal{V}} = [\![c]\!]_{\mathcal{M},\mathcal{V}} := c^{\mathcal{M}}$;
— pour tout $f \in \mathcal{S}_\mathtt{f}$ d'arité $n \geq 1$, pour tous termes $t_1, \ldots t_n$, $[\![f(t_1, \ldots, t_n)]\!]_{\mathcal{M},\mathcal{V}} := f^{\mathcal{M}}([\![t_1]\!]_{\mathcal{M},\mathcal{V}}, \ldots, [\![t_n]\!]_{\mathcal{M},\mathcal{V}})$.

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 3.28. La valeur $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}}$ d'une formule atomique φ est définie par :
— pour tout $R \in \mathcal{S}_{\mathrm{r}}$ d'arité n, pour tous termes $t_1, \ldots, t_n$,

$$[\![R(t_1, \ldots, t_n)]\!]_{\mathcal{M},\mathcal{V}} = 1 \qquad \text{ssi} \qquad ([\![t_1]\!]_{\mathcal{M},\mathcal{V}}, \ldots, [\![t_n]\!]_{\mathcal{M},\mathcal{V}}) \in R^{\mathcal{M}}.$$

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

Notation 3.29. 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 3.30 (Valeur d'une formule)**.** Etant donné un langage $\mathcal{S}$ et une formule φ de $\mathcal{F}_{\mathrm{po}}(\mathcal{S})$, la valeur de φ pour la $\mathcal{S}$-structure $\mathcal{M}$ et la valuation $\mathcal{V}$ est notée $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}}$ et est définie de la façon suivante :
— $[\![\varphi \wedge \psi]\!]_{\mathcal{M},\mathcal{V}} = 1$ ssi $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$ et $[\![\psi]\!]_{\mathcal{M},\mathcal{V}} = 1$;
— $[\![\varphi \vee \psi]\!]_{\mathcal{M},\mathcal{V}} = 1$ ssi $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$ ou $[\![\psi]\!]_{\mathcal{M},\mathcal{V}} = 1$;
— $[\![\neg\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$ ssi $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}} = 0$;
— $[\![\varphi \Rightarrow \psi]\!]_{\mathcal{M},\mathcal{V}} = 0$ ssi $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$ et $[\![\psi]\!]_{\mathcal{M},\mathcal{V}} = 0$;
— $[\![\forall x\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$ si et seulement si pour tout $a \in D_{\mathcal{M}}$, $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}[x:=a]} = 1$;
— $[\![\exists x\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$ si et seulement s'il existe $a \in D_{\mathcal{M}}$ tel que $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}[x:=a]} = 1$.

Remarque 3.31. Le quantificateur universel peut se considérer comme une sorte de grande conjonction. En fait, on a

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

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

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

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 $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$.

Remarque 3.32. La valeur de $[\![\forall x\varphi]\!]_{\mathcal{M},\mathcal{V}}$ ou $[\![\exists x\varphi]\!]_{\mathcal{M},\mathcal{V}}$ ne dépend pas de $\mathcal{V}(x)$. Par suite, la valeur de $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}}$, 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 $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}}$ ne dépend pas de $\mathcal{V}$ et par conséquent on écrira $[\![\varphi]\!]_{\mathcal{M}}$ à la place de $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}}$ et $\mathcal{M} \models \varphi$ à la place 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. 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

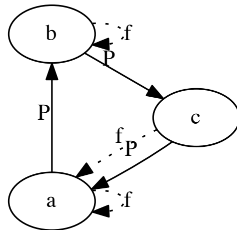


FIGURE 3.1 – Structure en forme de graphe étiqueté

— $\forall y \exists x p(x, y)$ 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 3.33. 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 3.1.

Pour calculer $[\![\, \varphi \,]\!]_{\mathcal{M},\mathcal{V}}$ on peut d'abord considérer toutes les valuations possibles de x et de y, soient 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 :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$[\![\, \exists y(P(x, f(y)) \vee P(y, f(x))) \,]\!]_{\mathcal{M},\mathcal{V}_a} = 1\,, \qquad \text{car } [\![\, P(x, f(y)) \vee P(y, f(x)) \,]\!]_{\mathcal{M},\mathcal{V}_{ab}} = 1.$$

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

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

$$[\![\, \exists y(P(x, f(y)) \vee P(y, f(x))) \,]\!]_{\mathcal{M},\mathcal{V}_b} = 1\,, \qquad \text{car } [\![\, P(x, f(y)) \vee P(y, f(x)) \,]\!]_{\mathcal{M},\mathcal{V}_{ba}} = 1.$$

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

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

$$[\![\, \exists y(P(x, f(y)) \vee P(y, f(x))) \,]\!]_{\mathcal{M},\mathcal{V}_c} = 1\,, \qquad \text{car } [\![\, P(x, f(y)) \vee P(y, f(x)) \,]\!]_{\mathcal{M},\mathcal{V}_{ca}} = 1.$$

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

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 3.34. 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 $a \in D_{\mathcal{M}}$; en plus, nous devons étendre l'interprétation, de façon que $c_a^{\mathcal{M}} = a$.

Vocabulaire

Définition 3.35 (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 3.36 (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 3.37 (Formule insatisfaisable)**.** Une formule close $\varphi \in \mathcal{F}_{\mathrm{po}}(\mathcal{S})$ est **insatisfaisable** si elle n'a pas de modèle.

Définition 3.38 (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 3.39 (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 3.40 (Equivalence)**.** Deux formules φ et ψ de $\mathcal{F}_{\mathrm{po}}(\mathcal{S})$ sont **équivalentes** (noté $\varphi \equiv \psi$) si pour toute $\mathcal{S}$-structure $\mathcal{M}$ et toute valuation $\mathcal{V} : X \to D_{\mathcal{M}}$, on $\mathcal{M}, \mathcal{V} \models \varphi$ ssi $\mathcal{M}, \mathcal{V} \models \psi$.

Attention : on parle d'équivalence de deux formules aussi quand elle 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 sot équivalentes ssi $\overline{\forall}(\varphi \Leftrightarrow \psi)$ est une tautologie. Ici, $\overline{\forall}(\varphi)$ est la clôture universelle de la formule φ, obtenue de φ en lui ajoutant une suite de quantificateurs universels $\forall x_1 \forall x_2 \ldots \forall x_n$, où $x_1, \ldots, x_n$ est la liste des variables libres de φ.

3.6 Manipulation de formules

3.6.1 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 $\psi = \varphi_{\{x \to t\}}$).

Remarque 3.41. 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 $\exists y R(x, y)$. La formule $\exists y R(y, y)$ n'est pas le résultat souhaité. On souhaite plutôt avoir comme résultat la formule suivante : $\exists z R(y, z)$, d'où on devine la nécessite de renommer les variables de façon qu'une substitution ne crée pas des nouvelle variables liées.

Si σ est la substitution $\{x_1 \to t_1, \ldots, x_n \to t_n\}$, on note φ_σ la formule φ dans laquelle toutes les occurrences libres de $x_1, \ldots, x_n$ ont été remplacées respectivement par $t_1, \ldots, t_n$. 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 $x \in Dom(\sigma)$ alors $t_\sigma = \sigma(x)$, (et $t_\sigma = x$ si $x \notin Dom(\sigma)$) ;
— si $t = c$ alors $t_\sigma = c$;
— si $t = f(t_1, \ldots, t_n)$ alors $t_\sigma = f(t_{1\sigma}, \ldots, t_{n\sigma})$.

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

$$\begin{aligned} R(t_1, \ldots, t_n)_\sigma &:= R(t_{1\sigma}, \ldots, t_{n\sigma})\,, \\ (\varphi \circ \psi)_\sigma &:= (\varphi_\sigma) \circ (\psi_\sigma), \text{ pour tout connecteur binaire } \circ\,, \\ (Qx.\varphi)_\sigma &:= \begin{cases} Qx.(\varphi_{\sigma[x \to x]})\,, & \text{si } x \notin Im(\sigma) \\ Qy.(\varphi_{\{x \to y\}})_\sigma\,, & \text{si } x \in Im(\sigma), \text{ où } y \notin Var(\varphi) \cup Dom(\sigma) \cup Im(\sigma)\,, \end{cases} \end{aligned}$$

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

Proposition 3.42. *Si $y \notin Var(\varphi)$, alors $\exists x \varphi \equiv \exists y(\varphi_{\{x \to y\}})$.*

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.

3.6.2 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 :

$$\neg\forall x \varphi \equiv \exists x \neg\varphi\,, \qquad \neg\exists x \varphi \equiv \forall x \neg\varphi\,.$$

— Lois de **distribution** des quantificateurs :

$$\forall x(\varphi \wedge \psi) \equiv (\forall x \varphi \wedge \forall x \psi)\,, \qquad \exists x(\varphi \vee \psi) \equiv (\exists x \varphi \vee \exists x \psi)\,.$$

— Lois de **permutation** des quantificateurs de même sorte :

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

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

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

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 $\exists x(\forall x F(x, y) \Rightarrow (G(x) \vee q))$, 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 $\{ x \to t \}$ la substitution qui remplace la variable x par la variable t. La loi de réalphabétisation peut se déduire de la regle élémentaire de réalphabétisation suivante :

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

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 3.43. 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 $[\![\exists x\neg\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$ et montrons que $[\![\neg\forall x\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$. De $\max_{a\in D_{\mathcal{M}}} [\![\neg\varphi]\!]_{\mathcal{M},\mathcal{V}[x:=a]} = 1$, nous déduisons que $[\![\neg\varphi]\!]_{\mathcal{M},\mathcal{V}[x:=a]} = 1$ pour un quelque $a \in D_{\mathcal{M}}$, donc $[\![\varphi]\!]_{\mathcal{M},\mathcal{V}[x:=a]} = 0$ et $[\![\forall x\varphi]\!]_{\mathcal{M},\mathcal{V}} = \min_{a\in D_{\mathcal{M}}} [\![\varphi]\!]_{\mathcal{M},\mathcal{V}[x:=a]} = 0$, donc $[\![\neg\forall x\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$.

(b) Supposons, par contre, que $[\![\neg\forall x\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$ et montrons que $[\![\exists x\neg\varphi]\!]_{\mathcal{M},\mathcal{V}} = 1$. On a bien que $[\![\forall x\varphi]\!]_{\mathcal{M},\mathcal{V}} = 0$; depuis

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

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

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 construire aussi un tel 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 l'existance 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 l'existence, seule la formule $\exists x\neg\varphi \Rightarrow \neg\forall x\varphi$ est considérée une tautologie, la formule $\neg\forall x\varphi \Rightarrow \exists x\neg\varphi$ n'est pas considérée une tautologie. Voir par exemple [Miq05].

Exercice 3.44. 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 $\exists x\varphi \wedge \exists x\psi$.

3.6.3 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.

3.6.3.1 Forme prénexe

Définition 3.45 (Forme prénexe)**.** Une formule φ est sous **forme prénexe** lorsqu'elle a la forme :

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \psi$$

où chaque Q_i est un quantificateur (existentiel ou universel) et ψ est sans quantificateurs. la partie $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n$ est appelée le **préfixe** et ψ étant qualifiée de **matrice**.

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

Théorème 3.46 (Forme prénexe équivalente)**.** *Pour toute formule $\varphi \in \mathcal{F}_{po}$ il existe une formule équivalente $\psi \in \mathcal{F}_{po}$ en forme prénexe.*

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

Etape 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 ;

Etape 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 3.47. 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 :

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

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

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

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

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

On ne peut ainsi appliquer la loi de passage :

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

3.6.3.2 Forme de Skolem

Définition 3.48. Une formule est sous **forme de Skolem** lorsqu'elle est sous forme prenexe et qu'elle ne contient que des quantifications universelles.

Exemple 3.49. 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 prenexe et on « supprime » les quantificateurs existentiels, en appliquant de façon itérée la règle suivante.

Règle de Skolémisation. Elle est de la forme

$$\frac{\exists y \psi}{\psi_{\{y \to f(z_1,\ldots,z_m)\}}}$$

où

— $z_1, \ldots, z_m$ 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 3.13). Une formule est *close* si elle ne contient pas de variables libres.

Exemple 3.50. Considérons une formule close $\varphi = \exists x \psi$ écrite sur un langage $\mathcal{S}$. La règle de skolémisation donne la formule $\psi_{\{x \to c\}}$, 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 $[\![\, \psi \,]\!]_{\mathcal{M},\mathcal{V}}$ 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 $\psi_{\{x \to c\}}$.

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

Exemple 3.51. 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 $[\![\, \psi \,]\!]_{\mathcal{M},[y:=b,x:=a]}$ ssi il existe une fonction $f : D_{\mathcal{M}} \to D_{\mathcal{M}}$ telle que, pour tout $b \in D_{\mathcal{M}}$, $[\![\, \psi \,]\!]_{\mathcal{M},[y:=b,x:=f(b)]}$.

Donc φ admet un modèle ssi $\forall y \psi_{\{x \to f(y)\}}$ a un modèle. Remarquez que la formule $\forall y \psi_{\{x \to f(y)\}}$ 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 3.52. Une formule universelle (c'est-à-dire une formule close contenant seulement des quantificateurs existentiels), obtenue par

— mise en forme prenexe, et ensuite
— application itérée de la règle de skolémisation

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

Exemple 3.53.

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

La skolémisation donne :

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

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 :

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

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 3.54. La version skolémisée 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 skolémisée est modèle de la formule initiale ;
- tout modèle de la formule initiale peut s'étendre en un modèle de la formule skolémisée, 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 3.55. *Si φ_s est obtenue par skolémisation à partir de φ alors φ_s est satisfaisable si et seulement si φ est satisfaisable.*

3.6.3.3 Forme clausale

Définition 3.56 (Forme clausale)**.** Une formule close est sous **forme clausale** si elle est

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

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

Définition 3.57.
- Un **littéral** est une formule atomique ou la négation d'une formule atomique.
- Soit ψ une formule avec $\mathsf{FV}(\psi) = \{ x_1, \ldots, x_n \}$. Sa **fermeture universelle** est la formule

$$\forall x_1 \ldots \forall x_n \psi \, .$$

- 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) equisatisfiable à la formule donnée.

Théorème 3.58 (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 3.59. 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 prénexe 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.

3.7 Unification

3.7.1 Substitions et MGUs

Dans la suite, soient $\mathcal{S}_\mathfrak{f}$ 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 3.60. Une **substitution** est une fonction $\sigma : X \to \mathcal{T}_{\mathcal{S}_\mathfrak{f}}(X)$ telle que l'ensemble $\{ x \in X \mid \sigma(x) \neq x \}$ est fini.

Exemple 3.61. Supposons $\mathcal{S}_\mathfrak{r} = \{ (f,2),(g,1) \}$ et $X = \{ x_0, \ldots, x_n, \ldots \}$. 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 implémentation 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 :

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

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 3.61, sera notée par

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

Exercice 3.62. 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 3.63. Pour tout terme t, on définit l'action de σ sur t comme suit :

$$x\,\sigma = \sigma(x) \,,$$
$$f(t_1, \ldots, t_n)\,\sigma = f(t_1\sigma, \ldots, t_n\sigma) \,.$$

Exemple 3.64. On a

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

Définition 3.65. La substitution **identité** (ou vide) est celle qui fixe toutes les variables. Elle est donc notée par []. La **composition** de deux substitutions σ et τ, notée $\tau \circ \sigma$, est la substitution définie par :

$$(\tau \circ \sigma)(x) = (x\,\sigma)\,\tau\,. \tag{3.2}$$

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

Exemple 3.66. Soient

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

Calculons $\tau \circ \sigma$:

$$\begin{array}{ccccc} & \sigma & & \rho & \\ \hline x & \mapsto & f(x,y) & \mapsto & f(g(y),f(x,y)) \\ y & \mapsto & y & \mapsto & f(x,z) \\ z & \mapsto & z & \mapsto & z \\ & & \vdots & & \end{array}$$

On a donc

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

Exercice 3.67. En généralisant l'exemple 3.66, 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 3.68. Prouvez les relations suivantes :

$$\begin{aligned} t\,[] &= t\,, \\ t\,(\tau \circ \sigma) &= (t\,\sigma)\,\tau\,. \end{aligned} \tag{3.3}$$

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

Remarque 3.69. 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 (3.2) et (3.3) qui, en renversant gauche et droite, pourraient apparaître à première vue un peu bizarres.

Définition 3.70. 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 3.71. Soit

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

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

Exemple 3.73.

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 $\mathsf{Unif}[\,(f(x,y),g(z))\,] = \varnothing$. De même, $\mathsf{Unif}[\,(x,g(x))\,] = \varnothing$.

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

Proposition 3.74. *Si* $\mathsf{Unif}[\,(s_1,t_1),\ldots,(s_n,t_n)\,] \neq \varnothing$, *alors il existe* $\sigma \in \mathsf{Unif}[\,(s_1,t_1),\ldots,(s_n,t_n)\,]$ *tel que* $\sigma \leq \tau$ *pour tout* $\tau \in \mathsf{Unif}[\,(s_1,t_1),\ldots,(s_n,t_n)\,]$.

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

Exemple 3.75. $\tau = [g(f(w))/x, g(f(w))/y] \in \mathsf{Unif}[\,(f(x,g(z)), f(g(z),y))\,]$, 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 $\rho = [f(w)/z]$.

3.7.2 Algorithme d'unification

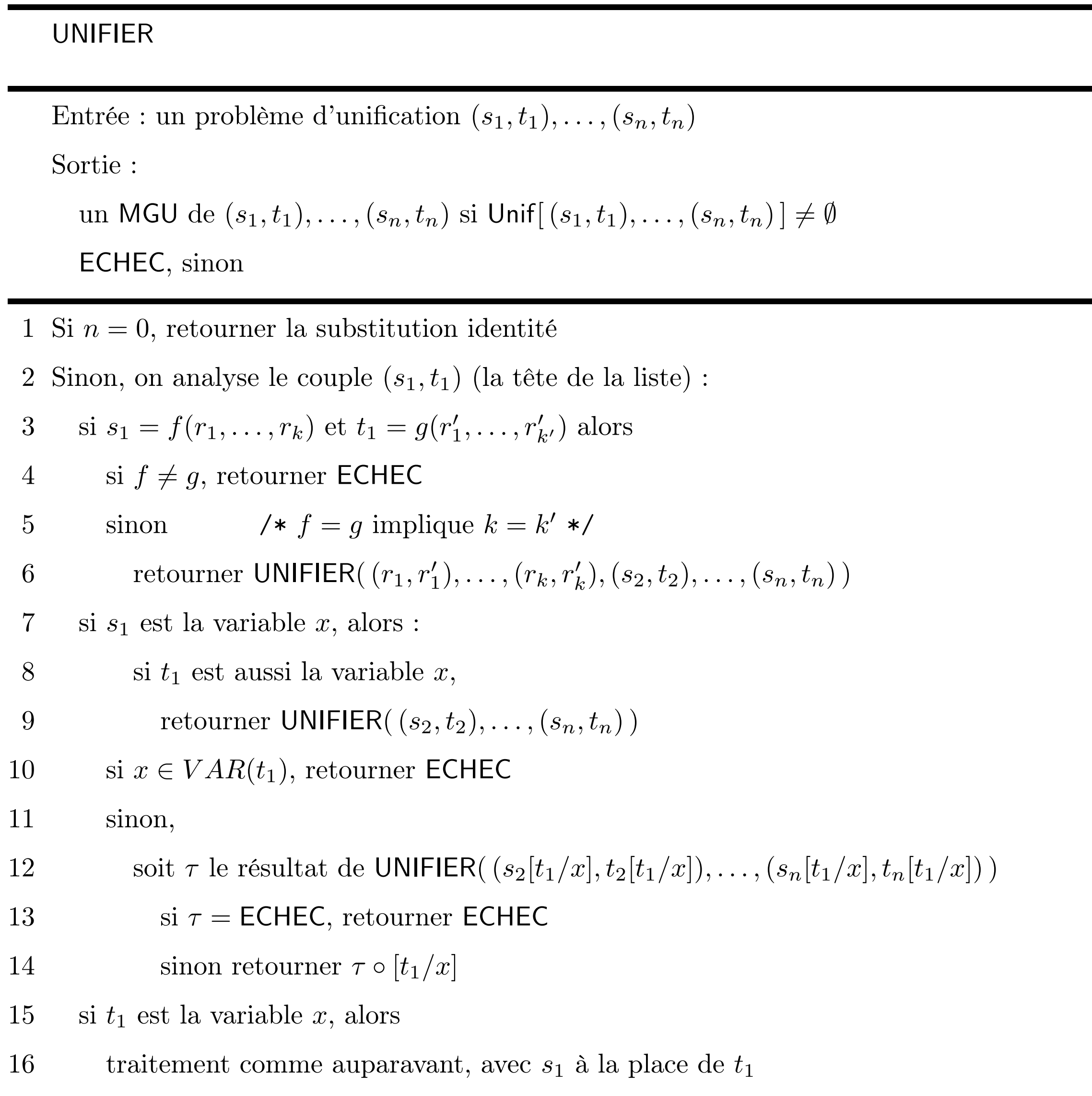
L'algorithme d'unification est illustré en figure 3.7.2. Nous donnons dans la suite des exemples de calcul de cet algorithme sur des problèmes d'unification.

Exemple 3.76. Considérez le problème suivant :
$$(f(x,g(z)), f(g(z),x))\,,\,(x,g(z))\,.$$
L'algorithme marche de la facon 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 3.77. 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. $(f(y,k(y),g(x)), f(k(x),k(y),y))$.



UNIFIER

Entrée : un problème d'unification $(s_1, t_1), \ldots, (s_n, t_n)$

Sortie :

un MGU de $(s_1, t_1), \ldots, (s_n, t_n)$ si $\mathsf{Unif}[\,(s_1, t_1), \ldots, (s_n, t_n)\,] \neq \emptyset$

ECHEC, sinon

```
1  Si n = 0, retourner la substitution identité
2  Sinon, on analyse le couple (s1, t1) (la tête de la liste) :
3    si s1 = f(r1, ..., rk) et t1 = g(r'1, ..., r'k') alors
4      si f ≠ g, retourner ECHEC
5      sinon          /* f = g implique k = k' */
6        retourner UNIFIER( (r1, r'1), ..., (rk, r'k), (s2, t2), ..., (sn, tn) )
7    si s1 est la variable x, alors :
8        si t1 est aussi la variable x,
9          retourner UNIFIER( (s2, t2), ..., (sn, tn) )
10     si x ∈ VAR(t1), retourner ECHEC
11     sinon,
12       soit τ le résultat de UNIFIER( (s2[t1/x], t2[t1/x]), ..., (sn[t1/x], tn[t1/x]) )
13         si τ = ECHEC, retourner ECHEC
14         sinon retourner τ ∘ [t1/x]
15   si t1 est la variable x, alors
16     traitement comme auparavant, avec s1 à la place de t1
```

FIGURE 3.2 – Algorithme d'unification

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

$$\sharp(x) = 1\,,$$
$$\sharp(f(t_1, \ldots, t_n) = 1 + \sum_{i=1,\ldots,n} \sharp t_i\,.$$

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

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

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.

3.7.3 Correction et completude

Nous souhaitons prouver les propositions suivantes.

Proposition 3.78 (Correction)**.** *Soit π un problème d'unification. Si* UNIFIER(π) *retourne* ECHEC*, alors* $\mathsf{Unif}[\,\pi\,] = \varnothing$ *; si* UNIFIER(π) *retourne une substitution σ, alors $\sigma \in \mathsf{Unif}[\,\pi\,]$ et, de plus, σ est un* MGU *de π.*

Proposition 3.79 (Complétude)**.** *Soit π un problème d'unification. Si* $\mathsf{Unif}[\,\pi\,] = \varnothing$*, alors* UNIFIER($\pi$) *retourne* ECHEC *; si* $\mathsf{Unif}[\,\pi\,] \neq \varnothing$*, alors* UNIFIER($\pi$) *retourne un* MGU *de π.*

Afin de prouver ces deux propositions, introduisons quelques notations qui sera utile, ainsi que la Proposition 3.80, qui est le résultat nécessaire le moins évidant à demontrer.

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

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

Avec cette notation remarquez que

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

Proposition 3.80. *Soient π et ψ deux problemes d'unification, soit σ un* MGU *de π. Alors*

$$\mathsf{Unif}[\,\pi,\psi\,] = \{\,\rho \circ \sigma \mid \rho \in \mathsf{Unif}[\,\psi_\sigma\,]\,\}\,. \tag{3.4}$$

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

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

Nous avons montré que l'ensemble sur la gauche de (3.4) est inclus dans celui de droite. Montrons donc l'autre inclusion. Si $\rho \in \mathsf{Unif}[\,\psi_\sigma\,]$, alors

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

donc $\rho \circ \sigma \in \mathsf{Unif}[\,\pi,\psi\,]$. Cela complète la preuve de l'égalité (3.4).

Soient maintenant ρ un MGU de ψ_σ et soit $\tau \in \mathsf{Unif}[\,\pi,\psi\,]$. A cause l'égalité (3.4), nous pouvons écrire $\tau = \rho' \circ \sigma$ avec $\rho' \in \mathsf{Unif}[\,\psi_\sigma\,]$; 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. □

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

Lemme 3.81. *Les faits suivants sont vrais :*

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

Lemme 3.82. *On a*

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

Lemme 3.83. *Un* MGU *de*

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

est ρ, *où* ρ *est un* MGU *de*

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

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

Lemme 3.84. *Supposons* $x \notin Var(t)$. *Un* MGU *de*

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

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

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

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

Les Lemmes 3.83 et 3.84 sont un conséquence de la Proposition 3.80, en raison du fait que
— la substitution identité [] est un MGU du problème (x,x) ;
— $[t/x]$ est évidemment un MGU du problème (x,t) quand $x \notin Var(t)$.

Exercice 3.85. A l'aide des Lemmes 3.81-3.84 complétez une preuve formelle de correction et complétude de l'algorithme d'unification.

Exemple : démonstration de la Proposition 3.78, correction de l'algorithme. L'algorithme retourne échec, sans appels recursifs, à cause des lignes 4,10 et 13. Pour les lignes 4 et 10, on utilise le Lemme 3.81. Pour la ligne 13, on utilise le Lemme 3.84.

Les appels recursifs se trouvent aux lignes 6 et 9. On justifie la ligne 6 par le Lemme 3.82, et la ligne 9 par le e Lemme 3.83.

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

La preuve que si UNIFIER(π) retourne σ, alors σ est un MGU de π est similaire. □

3.8 Résolution

3.8.1 Substitution, sur les formules propositionnelles

L'action d'une substitution s'étend aisément aux formules sans quantificateurs :
— $R(t_1, \ldots, t_n)\sigma = R(t_1\sigma, \ldots, t_n\sigma)$,
— $(\neg\varphi)\sigma = \neg(\varphi\sigma)$,
— $(\varphi \circ \psi)\sigma = \varphi\sigma \circ \psi\sigma$, $\circ \in \{ \vee, \wedge, \Rightarrow \}$.

Rappel (cf. Définition 3.57). 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

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

avec l_i des littéraux et $\{ x_1, \ldots, x_n \} = \mathsf{FV}(\{ l_1, \ldots, l_n \})$, 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. $R(t_1, \ldots, t_n)\sigma = R(t_1\sigma, \ldots, t_n\sigma)$,
2. $(\neg R(t_1, \ldots, t_n))\sigma = \neg(R(t_1, \ldots, t_n)\sigma)$,
3. $(\bigvee_{i=1}^{n} l_i)\sigma = \bigvee_{i=1}^{n} (l_i)\sigma$.

Notation 3.86. Si C est une clause universelle et σ une substitution, nous allons utiliser la notation $C\sigma$ pour la fermeture universelle de $C_{\mathsf{mat}}\sigma$. Évidemment, $C\sigma$ est aussi une clause universelle.

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

Lemme 3.87. *σ est un unificateur de l_0 et l_1 ssi*

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

Du Lemme il en découle tout de suite que l'ensemble des unificateurs de deux littéraux, appelons encore une fois $\mathsf{Unif}[\,l_0, l_1\,]$, 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.

3.8.2 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 :

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

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 3.88 (Règle de Résolution)**.** La suivante est une instance de la règle de résolution :

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

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

Exemple 3.89 (Règle de Factorisation)**.** La suivante est une instance de la règle de factorisation :

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

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

3.8.3 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, $i = 1, \ldots, n$ 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 subsitutions (et les quantificateurs universels), et l'autre étant la règle correspondante propositionnelle :

$$\frac{\dfrac{C \vee A_0}{C\sigma \vee A_0\sigma}\,\sigma \qquad \dfrac{C' \vee \neg A_1}{C'\sigma \vee \neg A_0\sigma}\,\sigma}{C\sigma \vee C'\sigma}\text{ Résolution propositionnelle}$$

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

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

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

Lemme 3.90. *Soient φ une formule sans quantificateurs, σ une substitution, $\mathcal{M}$ une $\mathcal{S}$-structure et $\mathcal{V}$ une valuation. On a que*

$$\mathcal{M}, \mathcal{V} \models \varphi\sigma \text{ ssi } \mathcal{M}, \mathcal{V}\sigma \models \varphi\,, \tag{3.5}$$

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

$$(\mathcal{V}\sigma)(x) = [\![\, \sigma(x) \,]\!]_{\mathcal{M},\mathcal{V}}\,.$$

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

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

Démonstration du Lemme 3.90. 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

$$[\![\, y \,]\!]_{\mathcal{M},\mathcal{V}\sigma} = (\mathcal{V}\sigma)(y) = [\![\, \sigma(y) \,]\!]_{\mathcal{M},\mathcal{V}}\,.$$

Ainsi, nous avons

$$\begin{aligned} \mathcal{M}, \mathcal{V} \models R(y_1, \ldots, y_m)\sigma &\text{ ssi } \mathcal{M}, \mathcal{V} \models R(\sigma(y_1), \ldots, \sigma(y_m)) \\ &\text{ ssi } ([\![\, \sigma(y_1) \,]\!]_{\mathcal{M},\mathcal{V}}, \ldots, [\![\, \sigma(y_m) \,]\!]_{\mathcal{M},\mathcal{V}}) \in R^{\mathcal{M}} \\ &\text{ ssi } ([\![\, y_1 \,]\!]_{\mathcal{M},\mathcal{V}\sigma}, \ldots, [\![\, y_m \,]\!]_{\mathcal{M},\mathcal{V}\sigma}) \in R^{\mathcal{M}} \\ &\text{ ssi } \mathcal{M}, \mathcal{V}\sigma \models R(y_1, \ldots, y_m). \end{aligned}$$

□

Lemme 3.91. *Pour toute clause universelle C et toute substitution σ, la règle d'inférence suivante est correcte :*

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

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 $\mathcal{M}, \mathcal{V} \models (C\sigma)_{\mathsf{mat}}$, où $\mathcal{V}$ est une valuation arbitraire. En considération que $(C\sigma)_{\mathsf{mat}} = (C_{\mathsf{mat}})\sigma$ et par le Lemme 3.90 cela revient à vérifier que $\mathcal{M}, \mathcal{V}\sigma \models C_{\mathsf{mat}}$; cette dernière relation et en effet vraie à cause de l'assomption $\mathcal{M} \models C$. □

Pour les règle de factorisation et résolution propositionnelles, nous devons les justifier. En fait, ces règle 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 3.92. *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 :*

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

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 $\mathcal{M}, \mathcal{V} \models (C_1 \vee C_2)_{\mathsf{mat}}$ pour toute valuation $\mathcal{V}$. Cela est une conséquence de $(C_1 \vee C_2)_{\mathsf{mat}} = (C_1)_{\mathsf{mat}} \vee (C_2)_{\mathsf{mat}}$, du fait que $\mathcal{M}, \mathcal{V} \models (C_1 \vee l)_{\mathsf{mat}} (= (C_1)_{\mathsf{mat}} \vee l)$, $\mathcal{M}, \mathcal{V} \models (C_2 \vee \neg l)_{\mathsf{mat}} (= (C_2)_{\mathsf{mat}} \vee \neg l)$, et du fait que la règle de la coupure propositionelle est correcte. □

Exercice 3.93. Montrez que la règle de factorisation propositionnelle s'étend aux clauses universelles.

3.8.4 Complètude 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 $\bot \notin \Gamma$.

Théorème 3.94. *Un ensemble saturé et cohérent de clauses admet au moins un modèle.*

En fait, nous allons montrer la proposition suivante :

Proposition 3.95. *Si un ensemble saturé de clauses Γ n'admet pas un modèle, alors $\bot \in \Gamma$.*

Démonstration. Soit $\mathcal{S} = (\mathcal{S}_{\mathtt{f}}, \mathcal{S}_{\mathtt{r}})$ 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) $v : \text{PROP} \to \{0, 1\}$, nous pouvons définir une $\mathcal{S}$-structure $\mathcal{M}_v$ de la façon suivante :

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

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

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

Lemme 3.96. *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 $D_{\mathcal{M}_v} = \mathcal{T}_{\mathcal{S}_\mathfrak{f}}(X)$ 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 $\mathcal{M}_v, \sigma \models C_{\mathsf{mat}}$, pour toute substitution σ. Rappelons que $[]$ est la substitution identité ; le Lemme 3.90 montre que $\mathcal{M}_v, \sigma \models C_{\mathsf{mat}}$ est équivalent à $\mathcal{M}_v, [] \models C_{\mathsf{mat}}\sigma$; cette condition revient à dire que $v(C\sigma) = 1$ (au sens propositionnel). □

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 $D_1, \ldots, D_n$ avec $D_n = \bot$ (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 factorization (propositionnelle).

Lemme 3.97. *Pour tout $i = 1, \ldots, n$, 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 $i = 1, \ldots, n$), 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 3.3.) 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 $D_k = C_k\sigma_k$.
   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 $\rho_j = [t_1/x_1, \ldots, t_n/x_n]$ et $\rho_k = [s_1/y_1, \ldots, s_m/y_m]$, alors $\tau = [t_1/x_1, \ldots, t_n/x_n, s_1/y_1, \ldots, s_m/y_m]$ 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 $\rho_j = \rho \circ \sigma$.
   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
   $$C_i\rho_i = [(C \vee \ell_0)\sigma]\rho_i = (C \vee \ell_0)(\rho_i \circ \sigma) = (C \vee \ell_0)\rho_j = C\rho_j \vee \ell_0\rho_j = D \vee \ell = D_j. \qquad \square$$

En particulier, le Lemme dit que $C_n \in \Gamma$ et que $C_n\rho_n = \bot$. Il est facile à voir que si $C_n \neq \bot$, alors $C_n\rho \neq \bot$ (pour n'importe quel ρ) ; par conséquent, nous avons $C_n = \bot \in \Gamma$. Cela complète la démonstration de la Proposition 3.95. □

Une analyse fine de la preuve de la Proposition 3.95 amène à une preuve du théorème suivant.

Théorème 3.98. *Si un ensemble de clauses Γ n'admet pas un modèle, alors il existe une preuve de $\bot$ à partir de Γ dans le calcul de la résolution.*

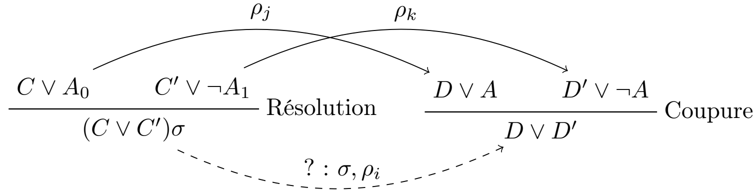


FIGURE 3.3 – Simulation de la coupure par la résolution

Exemple 3.99. 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 3.97, dans le calcul de la résolution, pour produire une preuve de la clause vide dans ce calcul. La figure 3.4 montre une preuve par coupure de la clause vide de Δ et sa simulation par résolution depuis Γ.

$$\dfrac{\dfrac{\dfrac{P(c) \vee Q(f(g(c))) \qquad \neg Q(f(g(c))) \vee Q(g(c))}{P(c) \vee Q(g(c))}\ \texttt{Coupure} \qquad \neg P(c)}{Q(g(c))}\ \texttt{Coupure} \qquad \neg Q(g(c))}{\bot}\ \texttt{Coupure}$$

$$\dfrac{\dfrac{\dfrac{P(x) \vee Q(f(y)) \qquad \neg Q(f(g(x))) \vee Q(g(x))}{P(x) \vee Q(g(x))}\ \texttt{Résolution}\ y \mapsto g(x) \qquad \neg P(c)}{Q(g(c))}\ \texttt{Résolution}\ x \mapsto c \qquad \neg Q(g(c))}{\bot}\ \texttt{Résolution}$$

FIGURE 3.4 – Simulation de preuves, exemple

3.8.5 Indecidabilité

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 considerer 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

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

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 3.100. *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.*

```
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 3.5 – Fichier d'entrée pour `Prover9`

Car décider de la satisfaisabilité d'un formule du premier ordre se réduit (via la mise en forme prenexe, la Skolemisation, et 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 3.101. *Il n'existe aucun algorithme tel que, étant donné un ensemble fini de clauses $\mathcal{C}$, il répond oui si $\mathcal{C}$ admet un modèle, et non si $\mathcal{C}$ est insatisfaisable.*

3.8.6 Utilisation d'un démonstrateur automatique

Exemple 3.102 (L'île misterieuse)**.** 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 debarque 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 3.5. 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 apparait dans la Figure 3.6. 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 assomptions) 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 !!! □

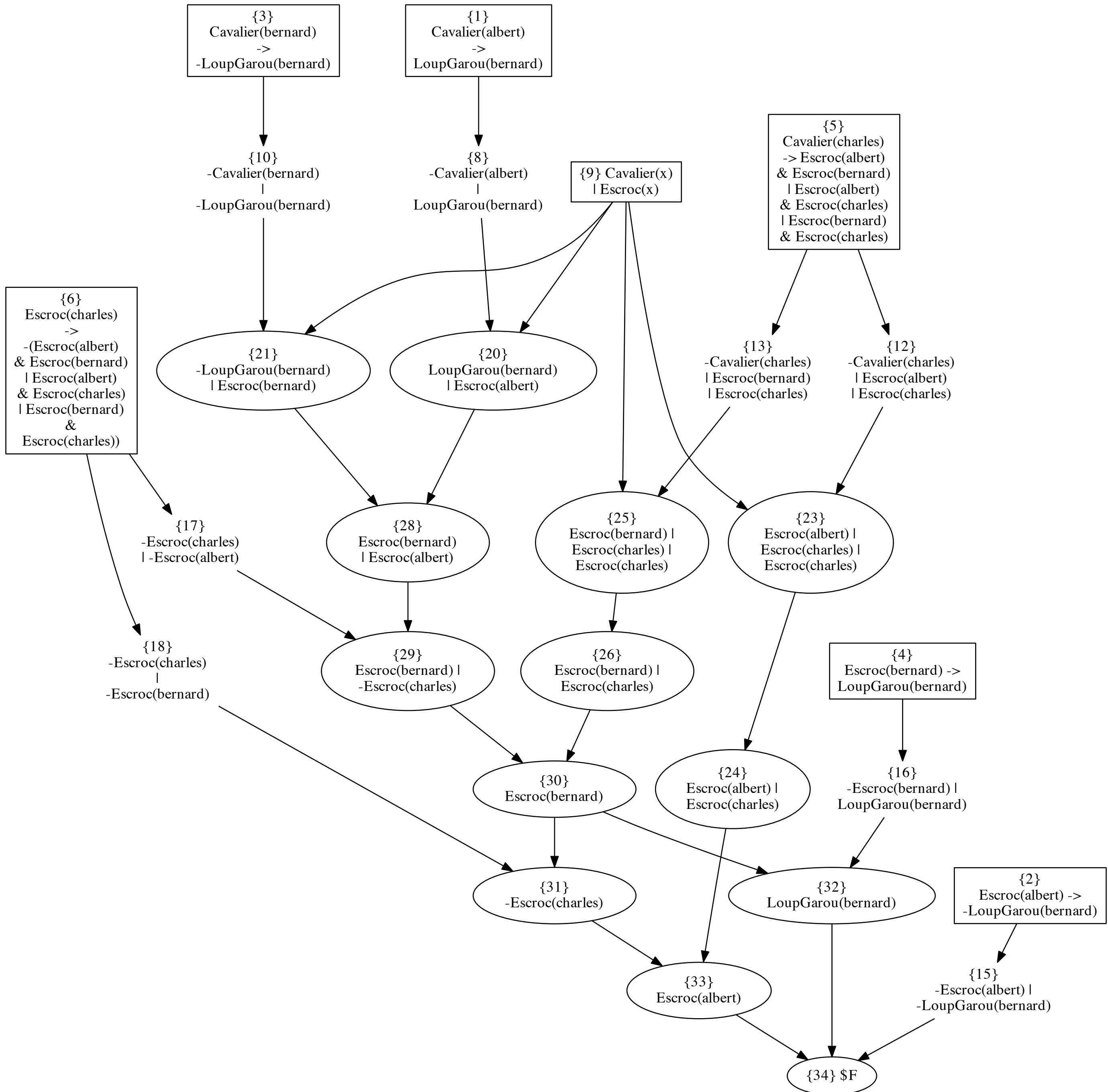


FIGURE 3.6 – Visualisation de la preuve construite par `Prover9`

```
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 3.7 – Fichier entrée pour le château Letot

Exemple 3.103 (Le mystère du Château Letot)**.** Écoutez cette autre histoire.

— *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 qui 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'*assomptions* ; la phrase 'Agate a tuée Agate' étant la formule *but* à démontrer [4]. Le code qui a été fourni en entrée à `Prover9` apparaît dans la figure 3.7. `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 assomptions, via sa négation (voir figure 3.8). La preuve que le but est une conséquence logique des assomptions apparaît dans la figure 3.9 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.

Un dernier remarque s'impose. Parmi les symboles de relation de notre langage nous avons utilisé le symbole = sans avoir ajouté, parmi les assomptions, aucune hypothèse sur ce symbole. 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

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

```
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 3.8 – Château Letot : forme clausale

symbole d'égalité et ajoute ces assomptions automatiquement. Car le traitement de l'égalité n'est pas optimal en utilisant la résolution seulement, on se sert aussi de la réglé de paramodulation, que nous présentons ci-dessous.

$$\frac{C \vee t_1 = t_2 \qquad D(t_3)}{(C \vee D(t_2))\sigma}\,\texttt{Paramodulation}$$

où σ est un unificateur de t_1 et t_3.

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

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

est une instance de la règle de paramodulation. □

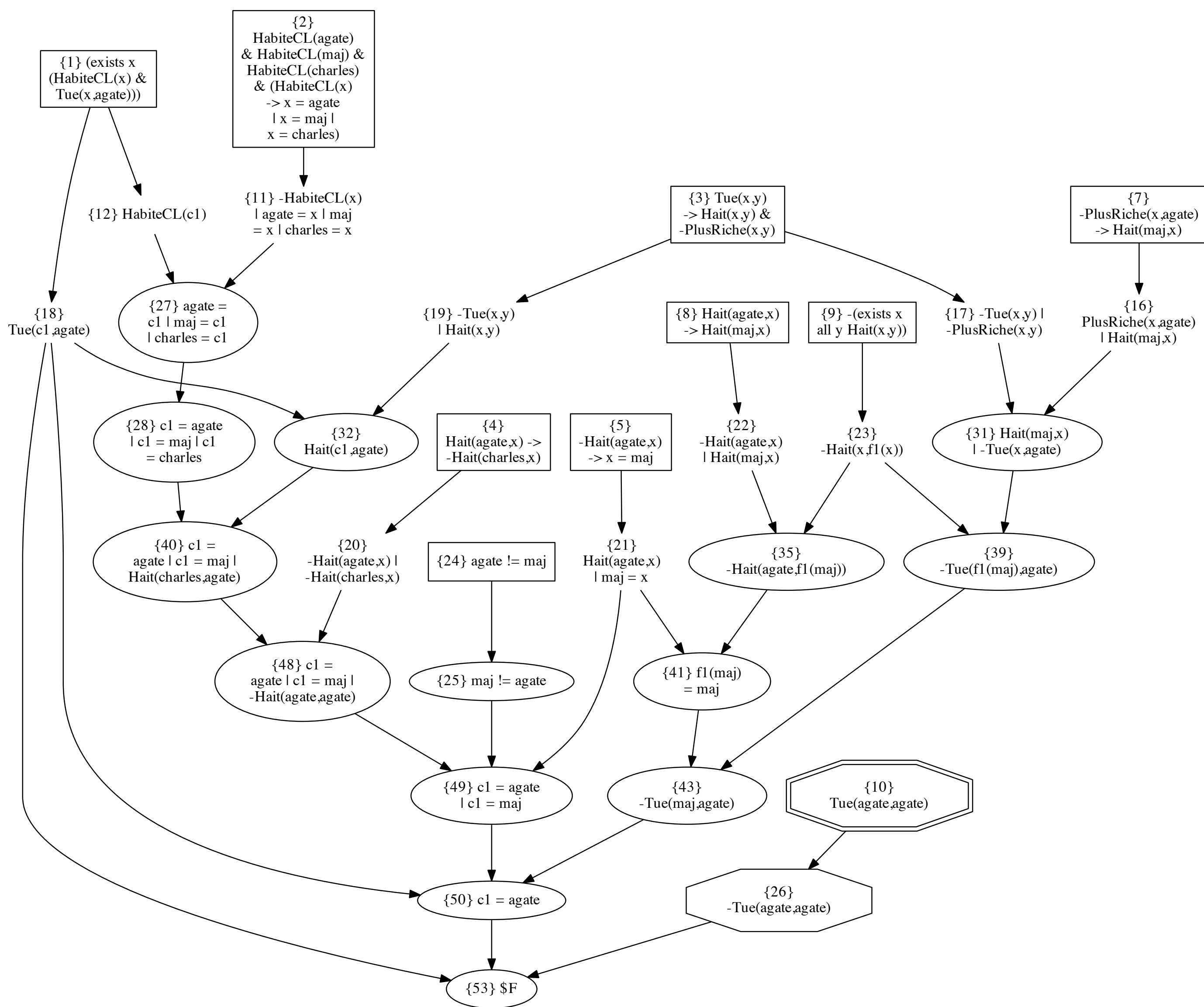


FIGURE 3.9 – La preuve trouvée par `Prover9`

Bibliographie

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

Chapitre 4

Calculabilité

Ce chapitre est largement inspiré du livre "Introductionà la calculabilité" de Pierre Wolper paru aux éditions Dunod [1].

4.1 Introduction

La question posée dans ce chapitre est de savoir quels problèmes sont solubles par un programme exécuté par un ordinateur.

Un *problème* est une fonction $f : I \rightarrow R$ d'un ensemble I d'instances vers un ensemble R de réponses possibles aux problème posé. Quand R est un ensemble binaire, on parle de *problème de décision* : la réponse attendue est *vrai* ou *faux*.

Quelques exemples de problèmes

1. Déterminer si un entier naturel est pair ou impair.
2. Trier un tableau de nombre.
3. Déterminer si un programme C s'arrête, quelles que soient les données fournies en entrée.
4. Déterminer si un polynôme à coefficients entiers a des racines entières (dixième problème de Hilbert).

On dira qu'un problème $f : I \rightarrow R$ est *calculable*, si il existe une procédure effective qui peut être appliquéeà n'importe instance $\iota \in I$ et qui a pour effet de produire le résultat $f(\iota)$.

Il se pose donc la question de définir la notion de *procédure effective* : elle doit être donnée par un ensemble fini d'instructions, utilisant des opérations très élémentaires, et doit pouvoir fournir le résultat en un nombre fini d'étapes. Par contre, on n'impose pas de limitation sur la taille de la mémoire disponible, ni sur la taille des données.

Cette présentation reste très informelle, il n'y a pas de définition rigoureuse de ce qu'est une procédure effective. Le but de ce cours est de présenter quelques *modèles* de cette notion en s'appuyant sur la thèse de Church-Turing.

Thèse de Church-Turing

1. Aucun modèle de la notion de fonction calculable n'est plus puissant que les machines de Turing.
2. Toute fonction calculable (au sens intuitif) est calculable par une machine de Turing.
3. Toutes les définitions formelles de calculabilité connuesà ce jour sont équivalentes.
4. Toutes les formalisations de la calculabilité établies par la suite seront équivalentes aux définitions connues.

Cette thèse est universellement reconnue comme vraie, bien que seul le troisième point peut être démontré.

Actuellement, on considère que les fonctions calculables sont celles calculées par un programme (peu importe le langage choisi).

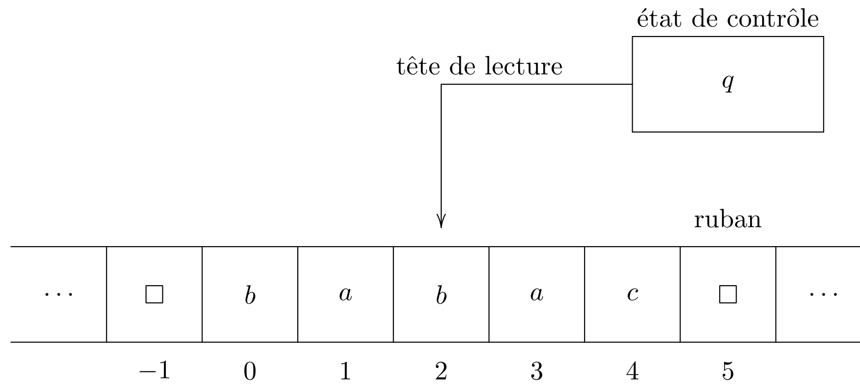


FIGURE 4.1 – Machine de Turing, intuitions

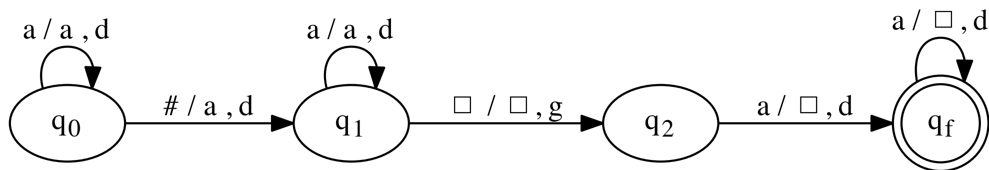


FIGURE 4.2 – Machine de Turing pour la somme

Nous présenterons dans ce cours deux notions de fonctions calculables : les fonctions calculables par une machines de Turing, les fonctions μ-calculables.

4.2 Machines de Turing

Définition 4.1. Une **machine de Turing** est une structure $T = \langle Q, \Gamma, \Sigma, \delta, q_0, B, F \rangle$, où

- — Q est un ensemble fini d'*états* ;
- — Γ est l'*alphabet de ruban* ;
- — $\Sigma \subseteq \Gamma$ est l'*alphabet d'entrée* ;
- — $q_0 \in Q$ est l'*état initial* ;
- — $F \subseteq Q$ est l'ensemble des *états accepteurs* (ou finaux) ;
- — $B \in \Gamma - \Sigma$ est le symbole blanc, souvent noté $\Box$;
- — $\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{ -1, +1 \}$ est la *fonction* (partielle) *de transition.*

Remarque 4.2. On peut lire la fonction de transition

$$\Delta(q, \sigma) = (q', \sigma', d)$$

(avec $d \in \{ -1, 1 \}$) par : *si on se trouve dans l'état q et la tête de lecture lit la lettre σ, alors on remplace σ par σ', on rentre dans l'état q', et on déplace la tête sur le ruban en direction d* (c'est-à-dire,à gauche si $d = -1$,à droite si $d = +1$).

Le ruban est comme un grand tableau indexé par les entiers, dont les cellules contiennent des lettres de Γ. On peut donc décrire l'état du ruban par une fonction $r : \mathbb{Z} \longrightarrow \Gamma$. Par ailleurs, on considère qu'à tout moment l'information dans le ruban soit finie, c'est-à-dire que tout l'ensemble des cases qui ne contiennent pas le symbole $\Box$ soit fini.

A tout moment d'un calcul, l'état global de la machine peut se décrire par le contenu du ruban, la position de la tête de lecture sur le ruban, et l'état de contrôle (voir la Figure 4.1). On peut formaliser la notion d'état global par la définition suivante, où "état global" devient "configuration".

Définition 4.3. Une **configuration** de T est un triplet (q, u, v) où :

— $q \in Q$ est l'état de contrôle ;
— $u \in \Gamma^*$ est le mot apparaîssant avant la position de la tête de lecture ;
— $v \in \Gamma^*$ est le mot apparaissant sur le ruban entre la position de la tête de lecture et le dernier caractère non blanc. Ce mot est donc fini et ne termine pas par $\square$.

Les configurations initiales sont toutes les configurations $c_\iota(w) = (q_0, \varepsilon, w)$, $w \in \Sigma^*$ Une *configuration* (q, u, v) est *finale* si $q \in F$.

Etant donnée une configuration (q, u, v), on peut toujours trouver $b \in \Gamma$ et $v' \in \Gamma^*$ tel que $v = bv'$ (Si $v = \varepsilon$, on pose $v' = \varepsilon$ et $b = \square$). De même, on pourra supposer que u se décompose en $u = u'b$.

Définition 4.4. Les **configurations dérivables** à partir de (q, u, v) sont définies de la façon suivante :

— si $\delta(q, b) = (q', a, +1)$, alors
$$(q, u, bw') \vdash_T (q', ua, w') \ ;$$
— si $\delta(q, b) = (q', a, -1)$, alors
$$(q, u'c, bw) \vdash_T (q', u', caw) \ .$$

On pose $(q, u, w) \vdash_T^* (q', u', w')$ s'il existe une suite de configurations $(q, u, w) = c_0, \ldots, c_n = (q', u', w')$ telle que $c_i \vdash_T c_{i+1}$, pour $i = 0, \ldots, n-1$.

Remarquez que, vue la définition donnée, une machine de Turing est déterministe : pour toute configuration c, il existe toujours au plus une configuration c' telle que $c \vdash_T c'$. De plus, une fois arrivée sur une configuration finale, la machine s'arrête.

Définition 4.5. On dit que T **s'arrête sur entrée** w (et on écrit cela par $T(w)\downarrow$) s'il existe une configuration finale c_f telle que $(q_0, \varepsilon, w) \vdash_T^* c_f$. Sinon, on dit que T **diverge sur entrée** w (et on écrit cela par $T(w)\uparrow$).

Langage reconnu par une machine de Turing. Le langage accepté par une machine de Turing est l'ensemble des mots $w \in \Sigma^*$ tels que

$$(q_0, \varepsilon, w) \vdash_T^* c_f \text{ où } c_f \text{ est une configuration finale.}$$

Le langage reconnu par une machine de Turing n'est pas reconnu par une procédure effective, en effet puisqu'on n'a pas interdit les calculs infinis, on ne peut pas toujours savoir si un mot est accepté ou pas.

Langage décidé par une machine de Turing. Un langage L est décidé par une machine de Turing T si T accepte L et T ne contient pas d'exécutions infinies.

Le langage décidé par une machine de Turing peut donc être reconnu par une procédure effective.

Définition 4.6. Un langage est dit **récursif** si il est décidé par une machine de Turing. Il est dit **récursivement énumérable** si il est accepté par une machine de Turing.

Fonction calculée par une machine de Turing.

Définition 4.7. Une machine de Turing calcule une fonction $f : \Sigma^* \to \Gamma^*$ si pour tout mot d'entrée w, elle s'arrête dans une configuration finale où $f(w)$ se trouve sur le ruban.

Une fonction est **calculable par une machine de Turing** si il existe une machine de Turing qui la calcule.

Exercice 4.8. Exécutez la machine de Turing de la figure 4.2 à partir de la configuration initiale $(q_0, \varepsilon, aa\sharp aaa)$. Que calcule, en général, cette machine ?

4.3 Problèmes de décision

On rappelle qu'un *problème de décision* est une fonction $P : I_P \to \{ Oui, Non \}$. On dit que I_P est l'ensemble des *instances de P*.

Exemple 4.9. `SAT` est le problème de décision suivante : une instance de `SAT` (donc un élément de $I_{\texttt{SAT}}$) est un ensemble fini de clauses ; pour un tel ensemble de clauses $\mathcal{C}$, on aura $\texttt{SAT}(\mathcal{C}) = Oui$ si $\mathcal{C}$ est satisfaisable, et $\texttt{SAT}(\mathcal{C}) = NON$ sinon.

En principe, tous les problèmes peuvent se réduireà des problèmes de décision. Considérons, par exemple, le problème de trouver le nombre chromatique d'un graphe. Soit C_n le problème de décision, dont les instances sont les graphes non-orientés, et tel que $C_n(V,E) = Oui$ ssi (V,E) possède un coloriage avec n couleur. Pour trouver le nombre chromatique d'un graphe, nous pouvons résoudre les problèmes $C_1, \ldots C_k \ldots$ jusqu'à trouver une réponse *Oui*.

Un *codage* est une fonction injective qui permet de représenter les instances d'un problème comme des mots sur l'alphabet Σ d'une machine de Turing T. Nous allons supposer qu'il existe toujours un codage canonique des l'ensemble de réponses $\{ Oui, Non \}$ sur le langage d'une machine de Turing ; par conséquent, nous ne ferons pas mention de pas ce codage dans la suite.

Définition 4.10. Un problème de décision P est **décidable** s'il existe une machine de Turing T et un codage $\gamma : I_P \to \Sigma^*$ telle que :

1. $T(w)\downarrow$, pour tout $w \in \Sigma^*$;
2. pour toute instance $i \in I_P$, $T(\gamma(i)) = Oui$ ssi $P(i) = Oui$.

Attention, dans la définition précédente on ne peut pas omettre la première clause. Si on le fait, on obtient la définition de problème semi-décidable :

Définition 4.11. Un problème de décision P est **semi-décidable** s'il existe une machine de Turing T et un codage $\gamma : I_P \to \Sigma^*$ tels que, pour toute instance $i \in I_P$: $P(i) = Oui$ ssi $T(\gamma(i))\downarrow$ et $T(\gamma(i)) = Oui$.

La notion de semi-décidabilité est très différente, elle ne dit rien sur l'arrêt de la machine pour une instance i telle $P(i) = Non$: en fait, dans ce cas, on pourrait avoir $T(\gamma(i))\downarrow$ *ou bien* $T(\gamma(i))\uparrow$.

On peut aisément se convaincre de la caractérisation suivante :

Proposition 4.12. *Un problème de décision P est semi-décidable s'il existe une machine de Turing T et un codage $\gamma : I_P \to \Sigma^*$ tels que, pour toute instance $i \in I_P$,*
- *si $P(i) = Oui$ alors $T(\gamma(i))\downarrow$,*
- *si $P(i) = Non$ alors $T(\gamma(i))\uparrow$.*

Étant donné P, soit $\neg P$ le problème avec les mêmes instances de P tel que $(\neg P)(i) = Oui$ ssi $P(i) = Non$. On peut montrer la proposition suivante :

Proposition 4.13. *Si P et $\neg P$ sont semi-décidables, alors P est décidable.*

Observez que l'implication inverse, si P est décidable, alors P e $\neg P$ sont semi-décidables, est trivialement vraie.

4.4 Un problème indécidable

Le problème de l'`ARRÊT` a comme instances les couples (T, w) avec T une machine de Turing et w un mot sur l'alphabet Σ de la machine. $\texttt{ARRÊT}(T,w) = OUI$ si et seulement si la machine de Turing T s'arrête sur entrée w (ce que nous avons noté par $T(w)\downarrow$).

Proposition 4.14. *Le problème de l'`ARRÊT` n'est pas décidable.*

Démonstration. Par l'absurde : soit $\mathcal{N}$ une machine de Turing qui s'arrête toujours, et soit γ un codage, tels que, pour tout (T, w), on ait

$$T(w)\downarrow \quad \text{ssi} \quad \mathcal{N}(\gamma(T,w)) = Oui\,.$$

Soit Σ l'alphabet de $\mathcal{N}$; nous pouvons supposer que $\gamma(T, w)$ est de la forme $\alpha(T)\sharp\beta(w)$, où α est un codage des machines de Turing vers les mots sur l'alphabet $\Sigma \setminus \{\sharp\}$, et β est un codage des mots sur un langage avec un nombre arbitraire de symboles vers les mots sur $\Sigma \setminus \{\sharp\}$. On a donc :

$$T(w)\downarrow \quad \text{ssi} \quad \mathcal{N}(\alpha(T)\sharp\beta(w)) = Oui\,.$$

Construisons une machine de Turing $\mathcal{D}$ comme suit. $\mathcal{D}$ prend un mot w sur le ruban et il ajouteà la droite du mot w son codage $\beta(w)$, separé par le symbole $\sharp$. Le mot $w\sharp\beta(w)$ se trouveraà ce point sur le ruban. Ensuite $\mathcal{D}$ utilise $\mathcal{N}$ comme un sous-module, avec $w\sharp\beta(w)$ en entrée. Si $\mathcal{N}$ réponds *Oui*, alors $\mathcal{D}$ rentre dans une boucle infinie et ne s'arrête pas. Si $\mathcal{N}$ réponds Non, alors $\mathcal{D}$ rentre dans l'état final et s'arrête.

Maintenant, posons la question si $\mathcal{D}$, avec entrée $\alpha(\mathcal{D})$, s'arrête.

Si c'est le cas, c'est parce que $\mathcal{N}$, avec entrée $\alpha(\mathcal{D})\sharp\beta(\alpha(\mathcal{D}))$ a répondu Non ; par les hypothèses faites sur $\mathcal{N}$, cela est équivalentà dire que $\mathcal{D}$ avec entrée $\gamma(\mathcal{D})$ ne s'arrête pas (on a donc obtenu une contradiction).

Par ailleurs, si c'est ne pas le cas, c'est parce que $\mathcal{N}$, avec entrée $\alpha(\mathcal{D})\sharp\beta(\alpha(\mathcal{D}))$ a répondu *Oui* ; par les hypothèses sur $\mathcal{N}$, cela est équivalentà dire que $\mathcal{D}$ avec entrée $\alpha(\mathcal{D})$ s'arrête (on a donc une autre contradiction). □

Remarque 4.15. Que l'encodage $\gamma(T, w)$ soit de la forme $\alpha(T)\sharp\beta(w)$ est un raccourcis pour ne pas alourdir la preuve par des détails techniques, qui sont par ailleurs nécessaires. Soit $\mathcal{MDT}$ l'ensemble des Machine de Turing ; nous supposons en fait que :

1. il existe un alphabet fini Σ_0 tel que $\sharp \not\in \Sigma_0$;
2. nous disposons d'un encodage canonique $\alpha : \mathcal{MDT} \to \Sigma_0^*$,
3. nous disposons un encodage canonique $\gamma : \mathbb{N}^* \to \Sigma_0^*$;
4. l'alphabet de n'importe quelle machine de Turing est de la forme $\Sigma = \{0, \ldots, n\}$ pour un quelque $n \geq 0$;
5. si P est un problème de décision tel que $I_P \subseteq \Sigma^*$, alors on a pas besoin d'encoder les instances de P comme des mots sur un autre alphabet.

Clairement, les hypothèses (1)à (4) ne posent pas de problèmes (via un renommage du symbol $\sharp$). L'hypothèse (5) peut être affaiblie, en demandant que si $I_P \subseteq \Sigma^*$, alors encodage de I_P soit calculé par une machine de Turing.

Étant dit cela et en dénotant par Σ_T l'alphabet d'une machine de Turing T, la façon correcte de définir le problème de l'arrêt est la suivante :

$$\begin{aligned} I_{\texttt{ARRÊT}} &= \{\,\alpha(T)\sharp\beta(w) \in (\Sigma_0 \cup \{\,\sharp\,\})^* \mid T \text{ une machine de Turing, } w \in \Sigma_T^*\,\}\,, \\ \texttt{ARRÊT}(\alpha(T)\sharp\beta(w)) &= Oui \text{ ssi } T(w)\downarrow\ . \end{aligned}$$

Remarquons en autre que le probleme de l'arrêt est semi-décidable. En fait, le Théorème suivant établie qu'il existe une machine de Turing U (car elle est dite universelle), qui se comporte comme un système d'exploitation : étant donnée en entrée la description d'une machine de Turing, elle simule la machine donnée en entrée, et va produire comme résultat un codage du résultat.

Théorème 4.16. *Il existe une machine de Turing U, telle que, pour tout toute entrée de la forme $\alpha(T)\sharp\beta(w)$, on a :*

1. $T(w)\downarrow$ *ssi* $U(\alpha(T)\sharp\beta(w))\downarrow$ *;*
2. *si* $T(w)\downarrow$, *alors* $U(\alpha(T)\sharp\beta(w)) = \beta(T(w))$.

Corollaire 4.17. *Le problème de l'arrêt est semi-décidable.*

Car il suffit de construire une machine A, qui utilise la machine U comme un sous-routine : dès que U rentre dans un état final (et donc elle retourne en tant que sous-routine), A remplace le contenu du ruban par le codage de la réponse positive *Oui*.

4.5 Fonctions récursives

La classe des fonction récursive est une classe de fonctions partielles (possiblement totales) de la forme $f : \mathbb{N}^n \to \mathbb{N}^m$, définie comme la plus petite classe de fonctions (partielles) contenant certaines fonction élémentaires et fermée sous certains schémas de définition.

Nous allons présenter ces schémas. En définissant la valeur $f(x)$ d'une fonction partielle, nous pouvons utiliser d'autres fonctions partielles qui peuvent ne pas être eux-mêmes définie sur ces valeurs. Si c'est le cas, alors il est implicite que la valeur de $f(x)$ n'est pas défini. Si nous souhaitons dire explicitement que la valeur de $f(x)$ n'est pas défini, alors nous allons écrire $f(x) = \perp$.

4.5.1 Les fonctions primitives recursives

Les fonctions primitives récursives sont définiesà partir de fonctions de base, d'une règle de composition, et d'une règle de récursion.

Les fonction de base sont les suivantes :

— La fonction $\mathbf{0}()$ qui n'a pas d'argument et retourne toujours 0.

— Les fonctions de projection :

$$\pi_i^k(n_1, \ldots, n_k) = n_i$$

qui permettent de sélection un argument parmi k.

— La fonction successeur :

$$\sigma(n) = n + 1$$

On utilise aussi les schémas suivants :

Schéma de composition. Soit $f : \mathbb{N}^n \to \mathbb{N}^m$ et $g : \mathbb{N}^n \to \mathbb{N}^k$. Posons

$$h(x) = g(f(x)) .$$

On dit que $h : \mathbb{N}^n \to \mathbb{N}^k$ est définie par compositionà partir de f et g.

Schéma de duplication. Etant donné $f_i : \mathbb{N}^n \to \mathbb{N}^{n_i}$ avec $i = 1, \ldots, k$, posons

$$h(x) = (f_1(x), \ldots, f_n(x))$$

La fonction $h : \mathbb{N}^n \to \mathbb{N}^m$, avec $m = \sum_{i=1,\ldots,k} n_i$ est dite définie par duplication depuis les f_i.

Exemple 4.18. Si $f_1 = f_2$ est la fonction identité, alors la fonction diagonale $\Delta(x) = (x, x)$, qui duplique la valeur de x, est définie par duplication depuis f_1 et f_2.

Schéma de récursion primitive Soient $f : \mathbb{N}^{1+m} \to \mathbb{N}^m$ et $g : \mathbb{N}^n \to \mathbb{N}^m$. Posons :

$$h(0, \bar{y}) = g(\bar{y}) , \qquad h(x + 1, \bar{y}) = f(x, h(x, \bar{y})) .$$

On dit que $h : \mathbb{N}^{1+n} \to \mathbb{N}^m$ est définie par récursion primitive depuis f et g.

Définition 4.19. Les fonctions primitives récursives sont toutes les fonction de base, et toutes les fonctions obtenues à partir de fonctions primitives récursives de base par l'application des schémas de composition, de duplication, et de récursion primitive.

Les prédicats primitives récursifs sont les fonctions primitives récursivesà valeur dans $\{0, 1\}$.

4.5.1.1 Exemples

Constantes Toutes les fonctions constantes sont primitives récursives :

$$\mathbf{j}() = \sigma^j(\mathbf{0}())$$

Addition

$$\begin{aligned} plus(n_1, 0) &= n_1 \\ plus(n_1, n_2 + 1) &= \sigma(plus(n_1, n_2)) \end{aligned}$$

Produit

$$\begin{aligned} produit(n,0) &= 0 \\ produit(n,m+1) &= plus(n, produit(n,m)) \end{aligned}$$

Prédecesseur

$$\begin{aligned} pred(m) &= 0 \text{ si } m = 0 \\ pred(m) &= m-1 \text{ si } m > 0 \end{aligned}$$

est primitive récursive puisqu'elle peut s'écrire par récursion primitive :

$$\begin{aligned} pred(0) &= 0 \\ pred(m+1) &= m \end{aligned}$$

Soustraction

$$\begin{aligned} moins(n,m) &= 0 \text{ si } n \leq m \\ moins(n,m) &= n-m \text{ sinon} \end{aligned}$$

est primitive récursive puisqu'elle peut s'écrire par réxcursion primitive :

$$\begin{aligned} moins(n,0) &= n \\ moins(n,m+1) &= pred(moins(n,m)) \end{aligned}$$

Signe

$$\begin{aligned} sg(n) &= 0 \text{ si } n = 0 \\ sg(n) &= 1 \text{ sinon} \end{aligned}$$

est primitive récursive puisqu'elle peut s'écrire par récursion primitive :

$$\begin{aligned} sg(0) &= 0 \\ sg(n+1) &= 1 \end{aligned}$$

Comparaisons Les fonctions caractéristiques des prédicats $<$, $=$, $\leq$ sont également primitifs récursifs :

$$\begin{aligned} ppetit(n,m) &= 1 \text{ si } n < m, 0 \text{ sinon} \\ egal(n,m) &= 1 \text{ si } n = m, 0 \text{ sinon} \end{aligned}$$

est primitive récursive puisqu'elle peut s'écrire par récursion primitive :

$$\begin{aligned} ppetit(n,m) &= sg(moins(m,n)) \\ egal(n,m) &= moins(1, sg(m-n) + sg(n-m)) \end{aligned}$$

La quantification bornée Si p est un prédicat primitif récursif, alors

$$\forall i \leq m, p(\bar{n}, i)$$
$$\forall i \leq m, p(\bar{n}, i)$$

sont des prédicats primitifs récursifs :

$$\begin{array}{rcl} \forall i \leq m, p(\bar{n}, i) & = & \Pi_{i=0}^{m} p(\bar{n}, i) \\ \exists i \leq m, p(\bar{n}, i) & = & sg(\Sigma_{i=0}^{m} p(\bar{n}, i)) \end{array}$$

Fonction conditionnelle Soit la fonction définie par :

$$\begin{array}{rcl} f(\bar{n}) & = & g_1(\bar{n}) \text{ si } p_1(\bar{n}) \\ & = & ... \\ & = & g_\ell(\bar{n}) \text{ si } p_\ell(\bar{n}) \end{array}$$

Si les f_i sont des fonctions primitives récursives et les p_i sont des prédicats primitifs récursifs alors f est une fonction primitive récursive :

$$f(\bar{n}) = \Sigma_{i=1}^{\ell} g_i(\bar{n}) \times p_i(\bar{n})$$

La minimisation bornée Si p est un prédicat primitif récursif, alors

$$\begin{array}{rcl} \mu i \leq m, p(\bar{n}, i) & = & \text{le plus petit } i \text{ tel que } p(\bar{n}, i) = 1 \\ & = & 0 \text{ siun tel } i \text{ n'existe pas} \end{array}$$

est une fonction primitive récursive :

$$\begin{array}{rcl} \mu i \leq 0, p(\bar{n}, i) & = & 0 \\ \mu i \leq (m+1), p(\bar{n}, i) & = & \mu i \leq m, p(\bar{n}, i) \text{ si } p(\bar{n}, m) = 0 \\ & = & m+1 \text{ si } p(\bar{n}, m+1) \text{ et } \neg\exists i \leq m+1, p(\bar{n}, i) \end{array}$$

4.5.1.2 Comparaison avec les fonctions calculables

Les fonctions primitives récursives sont toutes calculables par une procédure effective (au sens de la thèse de Turing-Church). Il existe des fonctions calculables qui ne sont pas primitives récursives. En effet, l'ensemble des fonctions primitives récursives est dénombrable, puisque chaque fonction peut être décrite par une cabine de caractères (sa description en fonctions de base et opérations)

Théorème 4.20. *Il existe des fonctions calculables qui ne sont pas primitives récursives.*

Démonstration. Les fonctions primitives récursives sont dénombrables. Soit donc $f_0, f_1, \ldots, f_n, \ldots$ une énumération des fonctions primitives récursives.

Nous utiliserons la notation suivante pour simplifier : $f_i(n) = f_i(n, \ldots, n)$. On considère le tableau suivant :

A	0	1	2	$\ldots$	j	$\ldots$
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$\ldots$	$f_0(j)$	$\ldots$
f_1	$f_1(0)$	$f_0(1)$	$f_1(2)$	$\ldots$	$f_1(j)$	$\ldots$
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$\ldots$	$f_2(j)$	$\ldots$
$\vdots$						
f_i	$f_i(0)$	$f_i(1)$	$f_i(2)$	$\ldots$	$f_i(j)$	$\ldots$
$\vdots$						

La case $A[i, j]$ du tableau contient donc l'entier naturel $f_i(j)$.
On définit maintenant la fonctionà 1 argument suivante :

$$g(n) = f_n(n) + 1 = A[n, n] + 1$$

Clairement, cette fonction n'est pas primitive récursive car elle ne peut être égaleà aucune des f_i. Par contre elle est calculable. Pour calculer $g(n)$, on procède de la faÃ§on suivante :

1. On énumère les fonctions primitives récursives jusqu'à f_n. Cette énumération peut se faire par un algorithme qui énumère systématiquement toutes les chaines de caractères et conservent celles qui décrivent une fonction primitive récursive.
2. ensuite on évalue $f_n(n)$, qui est calculable car f_n est primitive récursive
3. On évalue $f_n(n) + 1$

□

Un exemple célèbre de fonction calculable non primitive récursive est la fonction d'Ackermann :

$$\begin{aligned} Ack(0, m) &= m + 1 \\ Ack(k + 1, 0) &= Ack(k, 1) \\ Ack(k + 1, m + 1) &= Ack(k, Ack(k + 1), m) \end{aligned}$$

4.5.2 Fonctions μ-récursives

On les obtient comme les fonctions primitives récursives mais on autorise une opération supplémentaire : la minimisation non bornée.

La minimisation non bornée d'un prédicat $p(\bar{n}, i)$ généralise la minimisation bornée vue précédemment.

$$\begin{aligned} \mu i\, p(\bar{n}, i) &= \text{le plus petit } i \text{ tel que } p(\bar{n}, i) = 1 \\ &= 0 \text{ si un tel } i \text{ n'existe pas} \end{aligned}$$

La minimisation non bornée appliquéeà des prédicats primitifs récursifs ne produit pas que des fonctions calculables. En effet, prenons un prédicat $p(\bar{n}, i)$ et un $\bar{n}$ tel que $p(\bar{n}, i) = 0$ pour tout i. Il va falloir tester tous les i pour évaluer $\mu i\, p(\bar{n}, i)$, et donc la procédure ne terminera jamais. Il faut donc se limiter aux prédicats "sûrs" qui vont assurer que la fonction sera calculable :

Définition 4.21. Un prédicat $p(\bar{n}, i)$ est dit sûr si

$$\forall \bar{n}, \exists i\, p(\bar{n}, i).$$

Définition 4.22. Les fonctions et prédicats μ-recursifs sont ceux obtenusà partir des fonctions primitives récursives de base par

- composition
- duplication
- récursion primitive
- minimisation non bornée de prédicats sûrs.

Cette définition assure que les fonctions μ-récursives sont calculables. Remarquons qu'il n'existe pas de procédure effective pour décider qu'un prédicat est sûr.

Définition 4.23. La classe de fonction μ-récursives est la plus petite classe $\mathcal{R}$ de fonctions partielles telle que :

- toute fonction constante appartientà $\mathcal{R}$;
- toute projection appartientà $\mathcal{R}$;
- la fonction successeur s (avec $s(x) = x + 1$) appartient à $\mathcal{R}$;
- $\mathbb{R}$ sous le schémas de duplication, récursion primitive, minimisation.

Théorème 4.24. *Une fonction est calculable par une machine de Turing ssi elle est une fonction μ-récursive.*

Bibliographie

[1] Pierre Wolper. *Introduction à la calculabilité*. Sciences sup. Dunod, Paris, France, troisième edition, October 2006.