

INTRODUCTION À LA LOGIQUE

NOTES DE PRÉPARATION À L'AGRÉGATION D'INFORMATIQUE

SYLVAIN SCHMITZ

Université Paris Cité, CNRS, IRIF, France



© 2021–2022 Sylvain SCHMITZ
licensed under Creative Commons License CC-BY-SA


AVANT-PROPOS


Ces notes de préparation à l'agrégation d'informatique portent sur le programme de tronc commun de logique, qui est en fait le programme de logique des classes préparatoires filière scientifique en voie mathématiques, physique, informatique (MP2I/MPI) première et deuxième années¹. Ce programme comprend trois grandes parties :

- (1) syntaxe des formules logiques,
- (2) sémantique de vérité du calcul propositionnel,
- (3) déduction naturelle.

La liste de leçons de l'agrégation d'informatique² comporte une leçon en lien avec la logique, centrée sur la partie 2 du programme ci-dessus :

28. Formules du calcul propositionnel : représentation, formes normales, satisfiabilité. Applications.

Ces notes de préparation suivent globalement la structure du programme, ce qui est parfois atroce d'un point de vue pédagogique (introduire la logique du premier ordre avant la sémantique de la logique propositionnelle...), mais me semble préférable dans le cadre d'une préparation à l'agrégation, où la connaissance du programme est un point d'évaluation. On trouvera aussi dans ces notes quelques notions utiles au programme complémentaire de l'agrégation d'informatique pour les épreuves de fondements de l'informatique³; enfin, quelques compléments clairement hors-programme sont indiqués par une astérisque (et grisés dans la table des matières), tandis que des activités sur machine sont identifiées par le pictogramme .

Ces notes de préparation ne remplacent pas une lecture approfondie d'ouvrages; en particulier, l'épreuve d'oral de leçon nécessite de sélectionner des ouvrages sur lesquels construire sa leçon et ses développements. Des renvois vers des ouvrages de logiques sont systématiques dans ces notes, et identifiés dans la marge par le pictogramme .

Une bonne ressource sur le programme de classes préparatoires MP2I/MPI en général est le livre de Vincent BARRA intitulé *Informatique MP2I/MPI* et publié chez Ellipses en 2021; le programme de logique des classes préparatoires y est traité dans la partie V (chapitres 17 et 18). Il est cependant nécessaire pour l'agrégation d'aller un peu au-delà du programme de classes préparatoires (en particulier, tout ce qui explicitement mentionné comme « hors

1. https://cache.media.education.gouv.fr/file/SPE1-MEN-MESRI-4-2-2021/64/6/spe777_annexe_1373646.pdf, section 6

2. https://media.devenirenseignant.gouv.fr/file/agregation_externe_21/32/4/p2022_agreg_ext_informatique_lecons_1422324.pdf

3. https://media.devenirenseignant.gouv.fr/file/agregation_externe_21/22/3/p2022_agreg_ext_informatique_1414223.pdf

programme » en MP2I/MPI est au programme de l'agrégation). Je recommande particulièrement le livre de Jacques DUPARC (2015) pour débiter en logique (mais sans insister sur les aspects informatiques). Pour aller plus loin, les livres de Jean GOUBAULT-LARRECQ et Ian MACKIE (1997) et de René DAVID, Karim NOUR et Christophe RAFFALLI (2003) sont de bonnes références générales.

TABLE DES MATIÈRES

Partie 1. Syntaxe des formules logiques	1
1. Syntaxe	3
1.1. Signatures	3
1.2. Formules	4
1.3. Variables libres et variables liées	4
2. Sémantique	7
2.1. Interprétations	7
2.2. Sémantique	9
2.3. Modèles, satisfiabilité et validité	10
2.4. Exemples de formules	10
2.5. * Calculabilité et complexité	14
2.5.1. Cas général : structures potentiellement infinies	14
2.5.2. Le cas des structures finies	17
3. Substitutions	21
3.1. Lemme de substitution	22
3.2. α -renommages	22
3.3. Unification	24
3.4. * Algorithme d'unification par union-find	27
3.4.1. Treillis des substitutions	28
3.4.2. Un premier algorithme d'unification efficace	30
3.4.3. Algorithme d'unification de HUET	32
Partie 2. Sémantique de vérité du calcul propositionnel	35
4. Syntaxe	37
4.1. Arbres de syntaxe abstraite	37
4.2. Syntaxe concrète	38
5. Sémantique	39
5.1. Valeurs de vérité	39
5.1.1. Interprétations	39
5.1.2. Sémantique	39
5.1.3. Tables de vérité	41
5.1.4. * Étendre la syntaxe	42
5.1.5. * Complétude fonctionnelle	43
5.2. Satisfiabilité et validité	44
6. Conséquences et équivalences logiques	47
6.1. Conséquences logiques	47
6.2. Équivalences logiques	48
6.3. Substitutions propositionnelles	50

6.3.1.	Lemme de substitution propositionnelle	50
6.4.	Équivalences usuelles	51
6.5.	Compacité	52
6.5.1.	Compacité par le théorème de TYCHONOFF	52
6.5.2.	Compacité par arbres sémantiques et le lemme de KÖNIG	53
7.	Formes normales	55
7.1.	Forme normale négative	56
7.2.	Forme clausale	57
7.2.1.	Forme clausale logiquement équivalente	57
7.2.2.	Forme clausale équi-satisfiable	59
7.3.	Forme normale disjonctive	60
7.3.1.	Forme disjonctive complète	61
7.3.2.	* Forme disjonctive première	62
7.3.3.	* Forme normale disjonctive minimale	66
7.4.	* Diagrammes binaires de décision	69
7.4.1.	Formules de SHANNON ordonnées	70
7.4.2.	Formules de SHANNON complètes	70
7.4.3.	Formules de SHANNON réduites	71
7.4.4.	Lien avec les automates finis	72
8.	Le problème de satisfiabilité	73
8.1.	Le problème SAT	73
8.2.	Solveurs SAT	74
8.2.1.	☐ Format DIMACS	74
8.2.2.	☐ Utilisation d'un solveur SAT	74
8.3.	Recherche de modèle	75
8.3.1.	Recherche de modèle par énumération	75
8.3.2.	Recherche de modèle par simplification : l'algorithme de QUINE	77
8.3.3.	* Algorithme de DAVIS, PUTNAM, LOGEMANN et LOVELAND	81
8.4.	Application : résolution de dépendances logicielles	83
8.4.1.	☐ Modélisation sur un exemple	85
8.5.	Application : coloration de graphe	88
8.5.1.	☐ Modélisation sur un exemple	88
8.5.2.	Réduction dans le cas général	91
8.5.3.	* Théorème de DE BRUIJN et ERDŐS	92
Partie 3.	Déduction naturelle	95
9.	Déduction naturelle classique propositionnelle	97
9.1.	Exemples de dérivations en déduction naturelle	98
9.2.	Règles admissibles et variantes	100
9.3.	Correction et complétude	103
9.4.	Assistants de preuve	104
9.4.1.	☐ Exemples de preuves en Coq	105
9.4.2.	* Admissibilité des tactiques de Coq	107
9.4.3.	*☐ Règles de la déduction naturelle en Coq	108
10.	Déduction naturelle classique du premier ordre	113
10.1.	Eigenvariables	115
10.2.	* α -renommages	116

10.3.	Correction et complétude	118
11.	Recherche automatique de preuve en logique propositionnelle	121
11.1.	* Calcul des séquents propositionnel	122
11.2.	* Recherche de preuve en calcul des séquents propositionnel	124
11.2.1.	Propriété de branches finies	125
11.2.2.	Inversibilité et algorithme de recherche de preuve	126
11.3.	* Correction et complétude du calcul des séquents propositionnel	128
11.4.	* Traduction du calcul des séquents propositionnel en déduction naturelle	130
11.4.1.	Règles d'introduction à gauche en déduction naturelle	130
11.4.2.	Traduction des preuves de formules sous forme normale négative	131
11.4.3.	Règles de mise sous forme normale négative en déduction naturelle	132
11.4.4.	Complétude propositionnelle de la déduction naturelle	133
Références		135
	Quelques ouvrages sur la logique	135
	Quelques textes fondateurs	136
	Autres ouvrages	137
	Autres références	138

Partie 1

Syntaxe des formules logiques

Programme officiel. Le but de cette partie est de familiariser progressivement les étudiants avec la différence entre syntaxe et sémantique d'une part et de donner le vocabulaire permettant de modéliser une grande variété de situations (par exemple, satisfaction de contraintes, planification, diagnostic, vérification de modèles, *etc.*). L'étude des quantificateurs est l'occasion de formaliser les notions de variables libres et liées, et de portée, notions que l'on retrouve dans la pratique de la programmation.

Notions	Commentaires
Variables propositionnelles, connecteurs logiques, arité.	Notations : $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$.
Formules propositionnelles, définition par induction, représentation comme un arbre. Sous-formule. Taille et hauteur d'une formule.	Les formules sont des données informatiques. On fait le lien entre les écritures d'une formule comme mot et les parcours d'arbres.
Quantificateurs universel et existentiel. Variables liées, variables libres, portée. Substitution d'une variable.	On ne soulève aucune difficulté technique sur la substitution. L'unification est hors programme.

Mise en œuvre On implémente uniquement les formules propositionnelles sous forme d'arbres.

Le programme de logique introduit la syntaxe de la logique propositionnelle puis les quantificateurs universel et existentiel. S'agit-il ici de quantificateurs sur les propositions, ce qui mènerait à l'étude de QBF les formules booléennes quantifiées ? Le programme complémentaire de fondements de l'informatique indique pour sa part que la logique du premier ordre est au programme. Dans ces notes, nous allons donc plutôt préparer le terrain pour le programme complémentaire et introduire la syntaxe de la logique du premier ordre en section 1 – la syntaxe dans le cas propositionnel sera rappelée en section 4.

Autre étrangeté du programme : seule la sémantique du fragment propositionnel est au programme (voir partie 2) ; la sémantique de la logique du premier ordre n'est abordée qu'à un niveau intuitif dans la partie 3 sur la déduction naturelle. À nouveau, et en guise de préliminaire pour le programme complémentaire de fondements de l'informatique, dans ces notes nous allons aussi définir la sémantique de la logique du premier ordre en section 2 – la sémantique dans le cas propositionnel sera rappelée dans la section 5.

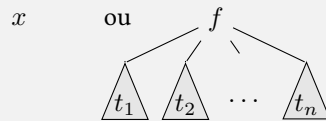
La notion de substitution de variable est l'objet d'une section à part (la section 3), puisqu'elle pose effectivement des difficultés techniques, et que le concept d' α -renommage sera

aussi utile pour le programme de fondements de l'informatique sur le λ -calcul. On y trouvera aussi en sections 3.3 et 3.4 deux algorithmes d'unification.

1. SYNTAXE

Résumé. Une *signature* du premier ordre $L = (\mathcal{F}, \mathcal{P})$ est constituée de symboles de fonction $f \in \mathcal{F}$ et de symboles de relation $R \in \mathcal{P}$. On associe une *arité* dans \mathbb{N} à chaque symbole et on note « \mathcal{F}_n » l'ensemble des symboles de fonction d'arité n et « \mathcal{P}_m » l'ensemble des symboles de relation d'arité m ; une fonction d'arité zéro est une *constante* et une relation d'arité zéro est une *proposition* ou « variable propositionnelle ».

Fixons une signature $L = (\mathcal{F}, \mathcal{P})$ et un ensemble infini dénombrable de *variables* X . L'ensemble $T(\mathcal{F}, X)$ des *termes* sur \mathcal{F} et X est l'ensemble des arbres de la forme



où $x \in X, n \in \mathbb{N}, f \in \mathcal{F}_n$ et t_1, t_2, \dots, t_n sont des termes.

Une *formule* est un arbre de la forme



où $m \in \mathbb{N}, R \in \mathcal{P}_m, t_1, t_2, \dots, t_m \in T(\mathcal{F}, X), x \in X$ et φ et ψ sont des formules. Une formule de la forme « $R(t_1, t_2, \dots, t_m)$ » est dite *atomique*.

Un terme sans variable est un *terme clos* et on note « $T(\mathcal{F})$ » pour l'ensemble des termes clos sur \mathcal{F} . Une variable x qui apparaît dans une formule mais pas sous un quantificateur $\exists x$ est dite *libre*; sinon elle est *liée*. On note « $\text{Libres}(\varphi)$ » pour l'ensemble des variables libres de φ et « $\text{Liées}(\varphi)$ » pour son ensemble de variables liées. Une formule sans variable libre est dite *close*.

1.1. Signatures. On définit tout d'abord une *signature* du premier ordre (aussi appelée un « langage du premier ordre ») comme une paire $L = (\mathcal{F}, \mathcal{P})$ formée de deux ensembles dénombrables \mathcal{F} de symboles de fonction et $\mathcal{P} \neq \emptyset$ de symboles de relation (aussi appelés « symboles de prédicat »), avec $\mathcal{F} \cap \mathcal{P} = \emptyset$, et tous deux munis d'une fonction d'arité $r: \mathcal{F} \cup \mathcal{P} \rightarrow \mathbb{N}$. On note $\mathcal{F}_n \stackrel{\text{def}}{=} \{f \in \mathcal{F} \mid r(f) = n\}$ et $\mathcal{P}_n \stackrel{\text{def}}{=} \{R \in \mathcal{P} \mid r(R) = n\}$ leurs

On peut aussi travailler avec des signatures non dénombrables, mais les preuves nécessitent alors d'utiliser des notions de théorie des ensembles comme le lemme de ZORN.

restrictions aux symboles d'arité n . Une fonction d'arité zéro est appelée une *constante*; une relation d'arité zéro est appelée une *proposition*, aussi appelée « variable propositionnelle ».

■ (DUPARC, 2015, sec. III.10.3),
(DAVID, NOUR et RAFFALLI, 2003,
sec. 1.2), (GOUBAULT-LARRECQ et
MACKIE, 1997, sec. 6.1),
(HARRISSON, 2009, sec. 3.1).

1.2. Formules. Soit X un ensemble infini dénombrable de symboles de variables. Les formules de la logique du premier ordre sur une signature $L = (\mathcal{F}, \mathcal{P})$ (aussi appelées « L -formules ») sont définies par la syntaxe abstraite

$$t ::= x \mid f(t_1, \dots, t_m) \quad (\text{termes})$$

$$\alpha ::= R(t_1, \dots, t_m) \quad (\text{formules atomiques})$$

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \quad (\text{formules})$$

où $x \in X$, $m \in \mathbb{N}$, $f \in \mathcal{F}_m$ et $R \in \mathcal{P}_m$. L'ensemble des termes sur X et \mathcal{F} est aussi dénoté $T(\mathcal{F}, X)$. Un *littéral* est une formule atomique α ou sa négation $\neg\alpha$.

On peut ajouter d'autres opérateurs booléens à cette syntaxe minimale. Alternativement, ces symboles peuvent être définis dans la logique :

$$\varphi \wedge \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \vee \neg\psi)$$

$$\forall x.\varphi \stackrel{\text{def}}{=} \neg\exists x.\neg\varphi \quad \varphi \rightarrow \psi \stackrel{\text{def}}{=} \neg\varphi \vee \psi$$

$$\top \stackrel{\text{def}}{=} \forall x.R(x, \dots, x) \vee \neg R(x, \dots, x) \quad \perp \stackrel{\text{def}}{=} \neg\top \quad \varphi \leftrightarrow \psi \stackrel{\text{def}}{=} \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$$

où R est un symbole arbitraire de \mathcal{P} (qui est supposé non vide).

Comme pour les formules propositionnelles, les formules de la logique du premier ordre sont donc des arbres de syntaxe abstraite. Par exemple, la figure 1.1 décrit la formule du buveur $\exists x.(B(x) \rightarrow \forall y.B(y))$. On a matérialisé dans cette figure les liens (en pointillés rouges) entre quantification et occurrences de chaque variable.

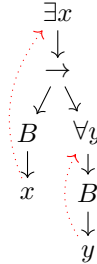


FIGURE 1.1. L'arbre de syntaxe abstraite de la formule du buveur $\exists x.(B(x) \rightarrow \forall y.B(y))$.

■ (DUPARC, 2015, sec. III.10.6),
(GOUBAULT-LARRECQ et
MACKIE, 1997, def. 6.3)

1.3. Variables libres et variables liées. L'ensemble $\text{Libres}(t)$ des *variables libres* (« *free variables* » en anglais) d'un terme t est défini inductivement par

$$\text{Libres}(x) \stackrel{\text{def}}{=} \{x\}, \quad \text{Libres}(f(t_1, \dots, t_m)) \stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq m} \text{Libres}(t_i).$$

Un terme t sans variable libre (i.e. $\text{Libres}(t) = \emptyset$) est dit *clos*; l'ensemble des termes clos est noté $T(\mathcal{F})$.

Les ensembles $\text{Libres}(\varphi)$ des *variables libres* et $\text{Liées}(\varphi)$ des *variables liées* (« *bound variables* » en anglais) d'une formule φ sont définis inductivement par

$$\begin{aligned} \text{Libres}(R(t_1, \dots, t_m)) &\stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq m} \text{Libres}(t_i), & \text{Liées}(R(t_1, \dots, t_m)) &\stackrel{\text{def}}{=} \emptyset, \\ \text{Libres}(\neg\varphi) &\stackrel{\text{def}}{=} \text{Libres}(\varphi), & \text{Liées}(\neg\varphi) &\stackrel{\text{def}}{=} \text{Liées}(\varphi), \\ \text{Libres}(\varphi \vee \psi) &\stackrel{\text{def}}{=} \text{Libres}(\varphi) \cup \text{Libres}(\psi), & \text{Liées}(\varphi \vee \psi) &\stackrel{\text{def}}{=} \text{Liées}(\varphi) \cup \text{Liées}(\psi), \\ \text{Libres}(\exists x.\varphi) &\stackrel{\text{def}}{=} \text{Libres}(\varphi) \setminus \{x\}, & \text{Liées}(\exists x.\varphi) &\stackrel{\text{def}}{=} \{x\} \cup \text{Liées}(\varphi). \end{aligned}$$

Une formule φ sans variable libre (c'est-à-dire telle que $\text{Libres}(\varphi) = \emptyset$) est dite *close*. Par exemple, la formule $\exists x.(B(x) \rightarrow \forall y.B(y))$ représentée dans la figure 1.1 est close. Si φ est le nom d'une formule qui n'est pas close, on note couramment « $\varphi(x_1, \dots, x_n)$ » pour expliciter que x_1, \dots, x_n est une énumération des variables de $\text{Libres}(\varphi) = \{x_1, \dots, x_n\}$, voir exemples 2.10 et 3.6 pour des exemples d'utilisation de cette notation.

2. SÉMANTIQUE

Résumé. Soit $L = (\mathcal{F}, \mathcal{P})$ une signature du premier ordre et $\mathbb{B} \stackrel{\text{def}}{=} \{\mathbf{F}, \mathbf{V}\}$ l'ensemble des *valeurs de vérité*, où \mathbf{F} désigne « faux » et \mathbf{V} désigne « vrai ». Une *interprétation* I d'une signature $(\mathcal{F}, \mathcal{P})$ est définie par un domaine D_I non vide, d'une fonction $f^I: D_I^n \rightarrow D_I$ pour tout $f \in \mathcal{F}_n$ et tout n et d'une relation $R^I: D_I^m \rightarrow \mathbb{B}$ pour tout $R \in \mathcal{P}_m$ et tout m .

- Étant donnée une *interprétation* I de L et une *valuation* $\rho: X \rightarrow D_I$ des variables,
 - la *sémantique* d'un terme t est un élément $\llbracket t \rrbracket_\rho^I \in D_I$ et
 - la *sémantique* d'une formule φ est une valeur de vérité $\llbracket \varphi \rrbracket_\rho^I \in \mathbb{B}$.

Dans les deux cas, ces sémantiques ne dépendent que de la valuation des variables libres de t et φ (propriété 2.6). On note « $I, \rho \models \varphi$ » si $\llbracket \varphi \rrbracket_\rho^I = \mathbf{V}$.

Une formule est *satisfiable* s'il existe une interprétation I et une valuation ρ telles que $I, \rho \models \varphi$. Une interprétation I est un *modèle* d'une formule φ (noté « $I \models \varphi$ ») si pour toute valuation ρ , on a $I, \rho \models \varphi$. Une formule φ est une *conséquence logique* d'un ensemble de formules S (noté « $S \models \varphi$ ») si, pour toute interprétation I et pour toute valuation ρ telles que $I, \rho \models \psi$ pour toute formule $\psi \in S$, on a $I, \rho \models \varphi$. Enfin, une formule φ est *valide* (noté « $\models \varphi$ ») si pour toute interprétation I et toute valuation ρ , on a $I, \rho \models \varphi$.

Fixons $L = (\mathcal{F}, \mathcal{P})$ une signature du premier ordre. On note $\mathbb{B} \stackrel{\text{def}}{=} \{\mathbf{F}, \mathbf{V}\}$ pour l'ensemble des *valeurs de vérité* (aussi appelé « algèbre de BOOLE »), où \mathbf{F} désigne « faux » et \mathbf{V} désigne « vrai », muni des opérations **non** : $\mathbb{B} \rightarrow \mathbb{B}$ et **ou** : $\mathbb{B}^2 \rightarrow \mathbb{B}$, définies par **non** $\mathbf{V} = \mathbf{F}$ ou $\mathbf{F} = \mathbf{F}$ et **non** $\mathbf{F} = \mathbf{V}$ ou $\mathbf{V} = \mathbf{V}$ ou $\mathbf{F} = \mathbf{F}$ ou $\mathbf{V} = \mathbf{V}$. Pour une famille $(b_j)_{j \in J}$ de valeurs de vérité $b_j \in \mathbb{B}$ indexée par un ensemble J , on définit aussi la disjonction distribuée par **Ou** $_{j \in J} b_j = \mathbf{V}$ si et seulement s'il existe $j \in J$ tel que $b_j = \mathbf{V}$.

■ (DAVID, NOUR et RAFFALLI, 2003, sec. 2.2),
(GOUBAULT-LARRECQ et MACKIE, 1997, sec. 6.2),
(HARRISSON, 2009, sec. 3.3).

2.1. Interprétations. Les formules du premier ordre sont interprétées sur des structures très générales, utilisables pour modéliser des structures mathématiques (ordres, groupes, corps, etc.) ou des structures relationnelles comme utilisées en bases de données.

Une *interprétation* I d'une signature $L = (\mathcal{F}, \mathcal{P})$ (aussi appelée une « L -structure ») est constituée d'un ensemble $D_I \neq \emptyset$, appelé son domaine ou son support, d'une fonction $f^I: D_I^{r(f)} \rightarrow D_I$ pour chaque $f \in \mathcal{F}$, et d'une relation $R^I: D_I^{r(R)} \rightarrow \mathbb{B}$ pour chaque $R \in \mathcal{P}$. Une telle interprétation est notée $I = (D_I, (f^I)_{f \in \mathcal{F}}, (R^I)_{R \in \mathcal{P}})$; dans le cas où \models^I est l'égalité sur D_I , on l'omet dans cette notation.

■ (DUPARC, 2015, sec. III.11.1)

▲ L'ensemble des formules valides change si l'on permet un domaine vide.

☞ En théorie des modèles et en théorie des modèles finis (CHANG et KEISLER, 1990; EBBINGHAUS, FLUM et THOMAS, 1994; LIBKIN, 2004), le symbole d'égalité est toujours présent et interprété comme l'égalité sur D_I ; on appelle de telles interprétations des *interprétations normales*.

Exemple 2.1 (interprétations propositionnelles). Posons $\mathcal{F} \stackrel{\text{def}}{=} \emptyset$ et $\mathcal{P} \stackrel{\text{def}}{=} \{P^{(0)}, Q^{(0)}\}$ où les exposants «⁽⁰⁾» indiquent des symboles d'arité 0. Dans cette signature, il n'y a pas de symbole de fonction, et il y a deux symboles de relation P et Q , tous deux d'arité 0. Une interprétation possible pour cette signature est I de domaine $D_I \stackrel{\text{def}}{=} \{\bullet\}$ avec $P^I \stackrel{\text{def}}{=} \mathbb{V}$ et $Q^I \stackrel{\text{def}}{=} \mathbb{F}$.

Exemple 2.2 (ordres). Posons $\mathcal{F} \stackrel{\text{def}}{=} \emptyset$ et $\mathcal{P} \stackrel{\text{def}}{=} \{<^{(2)}, =^{(2)}\}$. Dans cette signature, il n'y a pas de symboles de fonction, et il y a deux symboles de relation d'arité 2. Une interprétation $I = (\mathbb{Q}, <)$ pour cette signature est définie par $D_I \stackrel{\text{def}}{=} \mathbb{Q}$, $<^I \stackrel{\text{def}}{=} <$ l'ordre habituel sur les rationnels et $=^I$ l'égalité sur les rationnels.

Exemple 2.3 (arithmétique). Posons $\mathcal{F} \stackrel{\text{def}}{=} \{+^{(2)}, \times^{(2)}\}$ et $\mathcal{P} \stackrel{\text{def}}{=} \{=^{(2)}\}$. Cette signature comprend deux fonctions binaires $+$ et \times , et une relation binaire $=$. Une interprétation $I = (\mathbb{N}, +, \times)$ pour cette signature est définie par $D_I \stackrel{\text{def}}{=} \mathbb{N}$, $+^I : (n, m) \mapsto n + m$, $\times^I : (n, m) \mapsto nm$ et $=^I$ l'égalité sur \mathbb{N} .

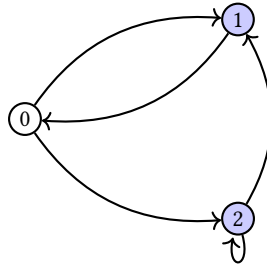


FIGURE 2.1. Un graphe fini orienté.

Exemple 2.4 (graphe fini orienté). Considérons le graphe fini orienté avec des sommets coloriés de la figure 2.1 et voyons comment le décrire comme une structure sur une signature appropriée. On pose pour cela $\mathcal{F} \stackrel{\text{def}}{=} \emptyset$ et $\mathcal{P} \stackrel{\text{def}}{=} \{E^{(2)}, =^{(2)}, B^{(1)}\}$ où E va représenter la relation d'adjacence du graphe et B indiquer pour chaque sommet s'il est bleu ou non. Le graphe de la figure 2.1 correspond alors à l'interprétation I de domaine $D_I \stackrel{\text{def}}{=} \{0, 1, 2\}$ avec $E^I \stackrel{\text{def}}{=} \{(0, 1), (0, 2), (1, 0), (2, 1), (2, 2)\}$, $B^I \stackrel{\text{def}}{=} \{1, 2\}$, et $=^I$ l'égalité entre sommets.

Exemple 2.5 (base de donnée relationnelle). Posons $L \stackrel{\text{def}}{=} \{\mathcal{F}, \mathcal{P}\}$ où

$$\mathcal{F} \stackrel{\text{def}}{=} \{shining^{(0)}, player^{(0)}, easyrider^{(0)}, apocalypsenow^{(0)}, kubrick^{(0)}, altman^{(0)}, \\ hopper^{(0)}, nicholson^{(0)}, robbins^{(0)}, coppola^{(0)}, champo^{(0)}, odeon^{(0)}\}$$

et

$$\mathcal{P} \stackrel{\text{def}}{=} \{Film^{(3)}, Seance^{(2)}, =^{(2)}\}.$$

Une interprétation I possible pour cette signature a pour domaine

$$D_I \stackrel{\text{def}}{=} \{ \textit{Shining}, \textit{The Player}, \textit{Easy Rider}, \textit{Apocalypse Now}, \text{KUBRICK}, \text{ALTMAN}, \text{HOPPER}, \\ \text{NICHOLSON}, \text{ROBBINS}, \text{HOPPER}, \text{Le Champo}, \text{Odéon} \}$$

où les constantes sont interprétées de manière évidente, et

$$\textit{Film}^I \stackrel{\text{def}}{=} \{ (\textit{Shining}, \text{KUBRICK}, \text{NICHOLSON}), (\textit{The Player}, \text{ALTMAN}, \text{ROBBINS}), \\ (\textit{Easy Rider}, \text{HOPPER}, \text{NICHOLSON}), (\textit{Easy Rider}, \text{HOPPER}, \text{HOPPER}), \\ (\textit{Apocalypse Now}, \text{COPPOLA}, \text{HOPPER}) \},$$

$$\textit{Seance}^I \stackrel{\text{def}}{=} \{ (\text{Le Champo}, \textit{Shining}), (\text{Le Champo}, \textit{Easy Rider}), (\text{Le Champo}, \textit{The Player}), \\ (\text{Odéon}, \textit{Easy Rider}) \}$$

et $=^I$ est l'égalité sur D_I . Cette interprétation correspond aux tables « Films » et « Séances » de la base de donnée ci-après.

titre	Films		Séances	
	réalisation	interprète	cinéma	titre
<i>Shining</i>	KUBRICK	NICHOLSON	Le Champo	<i>Shining</i>
<i>The Player</i>	ALTMAN	ROBBINS	Le Champo	<i>Easy Rider</i>
<i>Easy Rider</i>	HOPPER	NICHOLSON	Le Champo	<i>The Player</i>
<i>Easy Rider</i>	HOPPER	HOPPER	Odéon	<i>Easy Rider</i>
<i>Apocalypse Now</i>	COPPOLA	HOPPER		

☞ En général, une base de donnée relationnelle peut-être vue comme une interprétation de domaine fini sur une signature dotée des symboles de constantes adéquats, d'un symbole de relation par table, et du symbole d'égalité; on appelle cela une signature relationnelle.

2.2. Sémantique. Une *valuation* dans I est une fonction $\rho: X \rightarrow D_I$. On notera $\rho[e/x]$ pour la valuation qui associe e à x et $\rho(y)$ à $y \neq x$. Pour un terme t , sa sémantique $\llbracket t \rrbracket_\rho^I$ dans une interprétation I pour une valuation ρ est un élément de D_I défini inductivement par

$$\llbracket x \rrbracket_\rho^I \stackrel{\text{def}}{=} \rho(x), \quad \llbracket f(t_1, \dots, t_m) \rrbracket_\rho^I \stackrel{\text{def}}{=} f^I(\llbracket t_1 \rrbracket_\rho^I, \dots, \llbracket t_m \rrbracket_\rho^I)$$

pour tout m et tout $f \in \mathcal{F}_m$.

La sémantique $\llbracket \varphi \rrbracket_\rho^I$ d'une formule du premier ordre φ dans une interprétation I pour une valuation ρ est une valeur de vérité dans \mathbb{B} définie inductivement par

$$\llbracket R(t_1, \dots, t_m) \rrbracket_\rho^I \stackrel{\text{def}}{=} R^I(\llbracket t_1 \rrbracket_\rho^I, \dots, \llbracket t_m \rrbracket_\rho^I), \quad \llbracket \neg \varphi \rrbracket_\rho^I \stackrel{\text{def}}{=} \text{non } \llbracket \varphi \rrbracket_\rho^I, \\ \llbracket \varphi \vee \psi \rrbracket_\rho^I \stackrel{\text{def}}{=} \llbracket \varphi \rrbracket_\rho^I \text{ ou } \llbracket \psi \rrbracket_\rho^I, \quad \llbracket \exists x. \varphi \rrbracket_\rho^I \stackrel{\text{def}}{=} \text{Ou}_{e \in D_I} \llbracket \varphi \rrbracket_{\rho[e/x]}^I.$$

On dit que (I, ρ) *satisfait* φ , noté $I, \rho \models \varphi$, si $\llbracket \varphi \rrbracket_\rho^I = \mathbf{V}$; cette écriture peut être définie de manière équivalente par

$$\begin{array}{ll} I, \rho \models R(t_1, \dots, t_m) & \text{si } (\llbracket t_1 \rrbracket_\rho^I, \dots, \llbracket t_m \rrbracket_\rho^I) \in R^I, \\ I, \rho \models \neg \varphi & \text{si } I, \rho \not\models \varphi, \\ I, \rho \models \varphi \vee \psi & \text{si } I, \rho \models \varphi \text{ ou } I, \rho \models \psi, \\ I, \rho \models \exists x. \varphi & \text{si } \exists e \in D_I. I, \rho[e/x] \models \varphi. \end{array}$$

■ voir aussi (DUPARC, 2015, sec. III.11.3) pour une définition de la sémantique en termes de jeux

2.3. Modèles, satisfiabilité et validité. Une formule φ est *satisfiable* s'il existe une interprétation I et une valuation ρ telle que $I, \rho \models \varphi$. On dit que I est un *modèle* de φ , noté $I \models \varphi$, si pour toute valuation ρ , $I, \rho \models \varphi$. Une formule φ est *valide*, ce qui est noté $\models \varphi$, si pour toute interprétation I et toute valuation ρ , $I, \rho \models \varphi$; autrement dit, φ est valide si pour toute interprétation I , I est un modèle de φ .

Pour un ensemble de formules S , une interprétation I et une valuation ρ , on écrit $I, \rho \models S$ si $I, \rho \models \psi$ pour toutes les formules $\psi \in S$. Si de plus φ est une formule, on écrit $S \models \varphi$ si pour toute paire (I, ρ) telle que $I, \rho \models S$ on a $I, \rho \models \varphi$. Un ensemble S de formules est *insatisfiable* s'il n'existe pas I et ρ telles que $I, \rho \models S$, ou de manière équivalente si $S \models \perp$.

Notons que la satisfaction d'une formule ne dépend que de la valuation de ses variables libres.

Propriété 2.6. *Pour toute formule φ (resp. terme t), toute interprétation I , et toutes valuations ρ et ρ' telles que pour tout $x \in \text{Libres}(\varphi)$ (resp. $x \in \text{Libres}(t)$) $\rho(x) = \rho'(x)$, $\llbracket \varphi \rrbracket_{\rho}^I = \llbracket \varphi \rrbracket_{\rho'}^I$ (resp. $\llbracket t \rrbracket_{\rho}^I = \llbracket t \rrbracket_{\rho'}^I$).*

Démonstration. Par induction structurelle sur φ (resp. t). □

En particulier, par la propriété 2.6, φ avec $\text{Libres}(\varphi) = \{x_1, \dots, x_n\}$ est satisfiable si et seulement la formule close $\exists x_1 \dots \exists x_n. \varphi$ est satisfiable, et φ est valide si et seulement si la formule close $\forall x_1 \dots \forall x_n. \varphi$ est valide.

2.4. Exemples de formules. Voici quelques exemples de formules du premier ordre.

Exemple 2.7 (loi de PEIRCE). Prenons la signature $L = (\emptyset, \{P^{(0)}, Q^{(0)}\})$ de l'exemple 2.1. La *formule de PEIRCE* $((P \rightarrow Q) \rightarrow P) \rightarrow P$ est une formule du premier ordre sur cette signature. Elle est valide : pour toute interprétation I et pour toute valuation ρ , soit $P^I = \mathbf{V}$ et alors $I, \rho \models P$ et donc $I, \rho \models ((P \rightarrow Q) \rightarrow P) \rightarrow P$, soit $P^I = \mathbf{F}$ et alors $I, \rho \not\models P$ donc $I, \rho \models P \rightarrow Q$, donc $I, \rho \not\models (P \rightarrow Q) \rightarrow P$, et enfin $I, \rho \not\models ((P \rightarrow Q) \rightarrow P) \rightarrow P$.

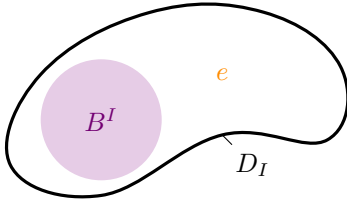
Exemple 2.8 (formule du buveur). Considérons la signature $L = (\emptyset, \{B^{(1)}\})$ avec une relation unaire B . La formule $\exists x.(B(x) \rightarrow \forall y.B(y))$, qui se lit habituellement « il existe un individu tel que s'il boit alors tout le monde boit », est valide.

Nous allons montrer que dans toute interprétation I et pour toute valuation ρ , $I, \rho \models \exists x.(B(x) \rightarrow \forall y.B(y))$. Cela revient à montrer que pour toute interprétation I , il existe un élément $e \in D_I$ tel que $I, \rho[e/x] \models B(x) \rightarrow \forall y.B(y)$, c'est-à-dire tel que

$$I, \rho[e/x] \not\models B(x) \quad \text{ou} \quad I, \rho[e/x] \models \forall y.B(y).$$

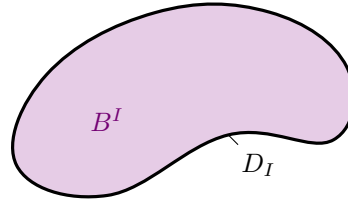
Il y a alors deux cas selon l'interprétation B^I du symbole de relation B :

☞ On notera que cette formule n'est pas valide si l'on permet un domaine d'interprétation vide. Elle n'est pas non plus valide en logique intuitionniste du premier ordre.



soit $B^I \subsetneq D_I$: il existe un élément « sobre » $e \in D_I$ tel que $e \notin B^I$, et on a bien

$$I, \rho[e/x] \not\models B(x)$$



soit $B^I = D_I$: comme $D_I \neq \emptyset$, on peut choisir n'importe quel $e \in D_I$ et on a bien

$$I, \rho[e/x] \models \forall y. B(y)$$

Exemple 2.9 (requêtes sur un graphe). Considérons à nouveau la signature et l'interprétation I de l'exemple 2.4 qui représente le graphe de la figure 2.1. On peut par exemple déterminer s'il existe un sommet de degré sortant un par la formule close

$$\exists x. (\exists y. E(x, y) \wedge \forall z. y = z \vee \neg E(x, z)) . \quad (2.1)$$

Cette formule va s'évaluer à \forall dans l'interprétation I et pour n'importe quelle valuation ρ (puisque la formule est close) : I est un donc modèle de la formule. En effet,

$$I, \rho \models \exists x. (\exists y. E(x, y) \wedge \forall z. y = z \vee \neg E(x, z))$$

$$\text{car } I, \rho[1/x] \models \exists y. E(x, y) \wedge \forall z. y = z \vee \neg E(x, z)$$

$$\text{car } I, \rho[1/x, 0/y] \models E(x, y) \wedge \forall z. y = z \vee \neg E(x, z)$$

car d'une part $I, \rho[1/x, 0/y] \models E(x, y)$ puisque $(1, 0) \in E^I$,

et d'autre part $I, \rho[1/x, 0/y] \models \forall z. y = z \vee \neg E(x, z)$:

en effet $I, \rho[1/x, 0/y, 0/z] \models y = z \vee \neg E(x, z)$ puisque y et z sont valués à 0

puis $I, \rho[1/x, 0/y, 1/z] \models y = z \vee \neg E(x, z)$ puisque $(1, 1) \notin E^I$

et enfin $I, \rho[1/x, 0/y, 2/z] \models y = z \vee \neg E(x, z)$ puisque $(1, 2) \notin E^I$.

Exemple 2.10 (requêtes sur une base de données). Considérons la signature et l'interprétation I de l'exemple 2.5. Une formule du premier ordre permet d'exprimer des requêtes sur la base de données correspondante : si x_1, \dots, x_n sont les variables libres de $\varphi(x_1, \dots, x_n)$, l'ensemble de n -uplets

$$\varphi(I) \stackrel{\text{def}}{=} \{(e_1, \dots, e_n) \mid I, [e_1/x_1, \dots, e_n/x_n] \models \varphi\}$$

est le résultat de la requête. Le langage SQL de requêtes sur une base de données permet d'exprimer des requêtes de la logique du premier ordre, et nous allons donner des exemples de requêtes en SQL pour chaque formule.

– Les titres des films présents dans la base de données :

$$\exists r \exists i. \text{Film}(x, r, i) \quad (2.2)$$

avec pour résultat $\{\text{Shining}, \text{The Player}, \text{Easy Rider}, \text{Apocalypse Now}\}$. Cette requête peut être exprimée en SQL sur la base de données de l'exemple 2.5 par

```
SELECT DISTINCT titre
FROM Films
```

☞ En bases de données, le calcul relationnel – qui est le nom que l'on donne à la logique du premier ordre sur une signature relationnelle – est équivalent à l'algèbre relationnelle – qui sont les opérations implémentées au sein des systèmes de gestion de base de données. Ce résultat connu comme le théorème de CODD apparaît dans le programme complémentaire d'informatique fondamentale, et peut être mentionné dans la leçon 23 « Requêtes en langage SQL ». ■ (JAUME et al., 2020, chap. 10), (ABITEBOUL, HULL et VIANU, 1995, chap. 5)

- Tous les cinémas qui diffusent un film avec HOPPER :

$$\exists t \exists r. \text{Film}(t, r, \text{hopper}) \wedge \text{Seance}(x, t) \quad (2.3)$$

avec pour résultat {Le Champo, Odéon}. La requête SQL correspondante est

```
SELECT DISTINCT cinema
FROM Films NATURAL JOIN Seances
WHERE Films.interprete = 'Hopper'
```

- Tous les cinémas qui diffusent un film de KUBRICK avec HOPPER :

$$\exists t. \text{Film}(t, \text{kubrick}, \text{hopper}) \wedge \text{Seance}(x, t) \quad (2.4)$$

avec pour résultat \emptyset . La requête SQL correspondante est

```
SELECT DISTINCT cinema
FROM Films NATURAL JOIN Seances
WHERE Films.interprete = 'Hopper'
AND Films.realisation = 'Kubrick'
```

- Les interprètes qui ont joué dans un film dirigé par KUBRICK ou par COPPOLA :

$$\exists t. \text{Film}(t, \text{kubrick}, x) \vee \text{Film}(t, \text{coppola}, x) \quad (2.5)$$

avec pour résultat {NICHOLSON, HOPPER}. La requête SQL correspondante est

```
SELECT DISTINCT interprete
FROM Films
WHERE Films.realisation = 'Kubrick'
OR Films.realisation = 'Coppola'
```

- Les réalisateurs qui ont joué dans un film qu'ils ont dirigé :

$$\exists t. \text{Film}(t, x, x) \quad (2.6)$$

avec pour résultat {HOPPER}. Une requête SQL correspondante est

```
SELECT DISTINCT F1.realisation
FROM Films F1 JOIN Films F2
ON F1.realisation = F2.interprete
AND F1.titre = F2.titre
```

- Les réalisateurs et les cinémas qui diffusent leurs films :

$$\exists t \exists i. \text{Film}(t, x, i) \wedge \text{Seance}(y, t) \quad (2.7)$$

avec pour résultat l'ensemble de paires {(KUBRICK, Le Champo), (ALTMAN, Le Champo), (HOPPER, Le Champo), (HOPPER, Odéon)}. Une requête SQL correspondante est

```
SELECT DISTINCT realisation, cinema
FROM Films NATURAL JOIN Seances
```

- Les réalisateurs dont les films passent dans tous les cinémas :

$$\forall c. (\exists t. \text{Seance}(c, t)) \rightarrow (\exists t \exists i. \text{Film}(t, x, i) \wedge \text{Seance}(c, t)) \quad (2.8)$$

avec pour résultat {HOPPER}. Une requête SQL possible pour cela est

```
SELECT DISTINCT F1.realisation
FROM Films F1
WHERE NOT EXISTS
  (SELECT S1.cinema
   FROM Seances S1
   EXCEPT
```

```

SELECT S2.cinema
FROM Seances S2 NATURAL JOIN Films F2
WHERE F1.realisation = F2.realisation)

```

À noter ici que par souci de lisibilité, on aurait pu définir une formule $cinema(c) \stackrel{\text{def}}{=} \exists t.Seance(c, t)$ et écrire notre requête (2.8) comme

$$\forall c.cinema(c) \rightarrow \exists t \exists i.Film(t, x, i) \wedge Seance(c, t) . \quad (2.9)$$

Comme la langue française est ambiguë, on aurait aussi pu comprendre cette requête comme

$$\exists t \exists i.Film(t, x, i) \wedge \forall c.cinema(c) \rightarrow Seance(c, t) \quad (2.10)$$

qui retourne aussi {HOPPER}.

- Les interprètes qui ont joué dans un film de HOPPER mais pas dans un film de KUBRICK :

$$(\exists t.Film(t, hopper, x)) \wedge \neg(\exists t.Film(t, kubrick, x)) \quad (2.11)$$

avec pour résultat {HOPPER}. Une requête SQL possible pour cela est

```

SELECT interprete
FROM Films
WHERE realisation = 'Hopper'
EXCEPT
SELECT interprete
FROM Films
WHERE realisation = 'Kubrick'

```

- Les interprètes qui n'ont joué que dans un seul film :

$$\exists t \exists r.Film(t, r, x) \wedge \forall t' \forall r'.Film(t', r', x) \rightarrow t = t' \quad (2.12)$$

avec pour résultat {ROBBINS}. Une requête SQL possible pour cela est

```

SELECT interprete
FROM Films
EXCEPT
SELECT F1.interprete
FROM Films F1, Films F2
WHERE F1.interprete = F2.interprete
AND F1.titre <> F2.titre

```

Exemple 2.11 (propriétés des ordres). Dans la signature de l'exemple 2.2, l'interprétation I de domaine \mathbb{Q} est un modèle de la formule

$$\forall x \exists y.y < x \quad (2.13)$$

qui exprime le fait que l'ordre n'est pas borné à gauche ; la formule est donc satisfiable. Il en serait de même si on avait pris $D_I \stackrel{\text{def}}{=} \mathbb{Z}$ et $<^I$ l'ordre sur les entiers relatifs. En revanche, si on avait pris $D_I \stackrel{\text{def}}{=} \mathbb{N}$, la formule (2.13) serait devenue fausse : cette formule n'est donc pas valide.

La formule close

$$\forall x \forall y.x < y \rightarrow \exists z.x < z \wedge z < y \quad (2.14)$$

exprime que l'ordre est dense. Elle est vraie sur \mathbb{Q} mais pas sur \mathbb{Z} et est donc satisfiable mais pas valide. La formule

$$\forall x \exists y \forall z. x < y \wedge \neg(x < z \wedge z < y) \quad (2.15)$$

est au contraire vraie sur \mathbb{Z} mais fausse sur \mathbb{Q} .

☞ La décidabilité et la complexité sont au programme MPI, section 9. Les exemples issus de la logique du premier ordre sont un peu avancés par rapport à ce programme, mais sont en revanche bien en lien avec le programme complémentaire de fondements de l'informatique et peuvent servir d'illustrations pour la leçon 27 « Décidabilité et indécidabilité. Exemples ».

2.5. * **Calculabilité et complexité.** La définition de la sémantique des formules du premier ordre ainsi que les définitions de validité et de satisfiabilité amènent à des questions algorithmiques naturelles : peut-on vérifier automatiquement si oui ou non une formule close s'évalue à \mathbf{V} dans une certaine interprétation ? Peut-on décider si une formule du premier ordre est satisfiable ? Si elle est valide ?

2.5.1. *Cas général : structures potentiellement infinies.* Dans le cas général, les structures sont *a priori* infinies (mais pas nécessairement). Le problème d'évaluation nécessite alors soit un moyen de décrire de manière finie l'interprétation sur laquelle on souhaite travailler, soit de fixer cette interprétation.

Évaluation et équation diophantiennes. Le problème d'ÉVALUATION DANS I où I est une interprétation fixée est le problème de décision suivant :

Problème (ÉVALUATION DANS I).

instance : une formule close φ

question : est-ce que $I \models \varphi$?

On va voir que ce problème est indécidable.

Une *équation diophantienne* est une équation $p(x_1, \dots, x_n) = 0$ où p est un polynôme à coefficients dans \mathbb{Z} pour laquelle on cherche une solution (x_1, \dots, x_n) dans \mathbb{Z}^n . Par exemple, $3x_1^2 - 2x_1x_2 - x_2^2x_3 - 7 = 0$ est une équation diophantienne qui a une solution $(1, 2, -2)$, tandis que $x_1^2 + x_2^2 + 1 = 0$ n'a pas de solution. Le problème de décision correspondant est connu comme « dixième problème de HILBERT ».

Problème (ÉQUATION DIOPHANTIQUE).

instance : une équation diophantienne $p(x_1, \dots, x_n) = 0$

question : existe-t'il une solution dans \mathbb{Z}^n ?

Une conséquence d'un résultat de MATIASSEVITCH en 1970 est qu'il n'existe pas d'algorithme capable de dire s'il existe au moins une solution à une équation diophantienne.

Théorème 2.12 (MATIASSEVITCH). *Le problème ÉQUATION DIOPHANTIQUE est indécidable.*

Une conséquence du théorème de MATIASSEVITCH est que les théories arithmétiques sur les entiers ou sur les naturels sont indécidables. Ici, on prend $\mathcal{F} = \{+(^{(2)}, \times^{(2)})\}$ et $\mathcal{P} = \{=(^{(2)})\}$ et on dénote par $(\mathbb{Z}, +, \times)$ l'interprétation sur les entiers de cette signature et par $(\mathbb{N}, +, \times)$ celle sur les naturels.

Corollaire 2.13 (indécidabilité de l'ÉVALUATION). *Pour $I = (\mathbb{Z}, +, \times)$ ou $I = (\mathbb{N}, +, \times)$, le problème d'ÉVALUATION DANS I est indécidable.*

Démonstration. Comme vu dans l'exemple 3.6, on peut exprimer par les formules $zero(z) \stackrel{\text{def}}{=} z + z = z$ et $un(u) \stackrel{\text{def}}{=} \neg zero(u) \wedge u \times u = u$ que les variables z et u soient valuées à 0 et 1 respectivement.

Étant donnée une équation diophantienne $p(x_1, \dots, x_n) = 0$, on peut l'écrire de manière équivalente comme $p_1(x_1, \dots, x_n) = p_2(x_1, \dots, x_n)$ où tous les coefficients de p_1 et p_2 sont dans \mathbb{N} . Par exemple, $3x_1^2 - 2x_1x_2 - x_2^2x_3 - 7 = 0$ s'écrit de manière équivalente comme $3x_1^2 = 2x_1x_2 + x_2^2x_3 + 7$.

On peut alors construire une formule universelle close

$$\varphi \stackrel{\text{def}}{=} \forall z \forall u. (zero(z) \wedge un(u)) \rightarrow \forall x_1 \dots \forall x_n. \neg (t_1(x_1, \dots, x_n, z, u) = t_2(x_1, \dots, x_n, z, u))$$

où t_1 et t_2 sont des termes qui représentent les polynômes p_1 et p_2 . Par exemple, $3x_1^2$ est représenté par $x_1 \times x_1 + x_1 \times x_1 + x_1 \times x_1$, et $2x_1x_2 + x_2^2x_3 + 7$ est représenté par $x_1 \times x_2 + x_1 \times x_2 + x_2 \times x_2 \times x_3 + u + u + u + u + u + u + u$. La formule φ est telle que $(\mathbb{Z}, +, \times) \models \varphi$ si et seulement si l'équation diophantienne $p(x_1, \dots, x_n) = 0$ n'a pas de solution. Par le théorème de MATIASSEVITCH, on ne peut donc pas décider si $(\mathbb{Z}, +, \times) \models \varphi$.

Pour le cas $I = (\mathbb{N}, +, \times)$, on fait le même raisonnement en observant que l'équation diophantienne $p(x_1, \dots, x_n) = 0$ a une solution dans \mathbb{Z}^n si et seulement si $p(y_1 - z_1, \dots, y_n - z_n) = 0$ a une solution dans \mathbb{N}^{2n} . \square

Satisfiabilité et problème de pavage du plan. Le problème de satisfiabilité pour la logique du premier ordre est le problème de décision suivant :

Problème (SATISFIABILITÉ).

instance : une formule φ

question : φ est-elle satisfiable ?

Dans cette section, nous allons voir que ce problème est indécidable. Nous allons exhiber pour cela une réduction depuis le problème de *pavage du plan*. L'entrée de problème est un *catalogue*, qui est un ensemble fini C de *tuiles* carrées avec une couleur par côté comme celles de la figure 2.2.

■ (BÖRGER, GRÄDEL et GUREVICH, 1997, sec. 3.1)

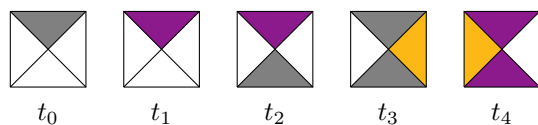


FIGURE 2.2. Un catalogue $C_{\text{ex}} = \{t_0, t_1, t_2, t_3, t_4\}$.

Le but pour un catalogue C donné est de déterminer s'il est possible de couvrir le plan $\mathbb{N} \times \mathbb{N}$ en respectant les couleurs. On peut pour cela réutiliser les tuiles du catalogue, mais celles-ci ne peuvent pas être tournées. La figure 2.3 montre deux exemples de pavages possibles avec le catalogue de la figure 2.2. Formellement, un catalogue C est associé à deux relations binaires $H \subseteq C \times C$ de contraintes horizontales et $V \subseteq C \times C$ de contraintes verticales, où $(t, t') \in H$ si la couleur de droite de t est la même que la couleur de gauche de t' et $(t, t') \in V$ si la couleur du haut de t est la même que la couleur du bas de t' . Par exemple,

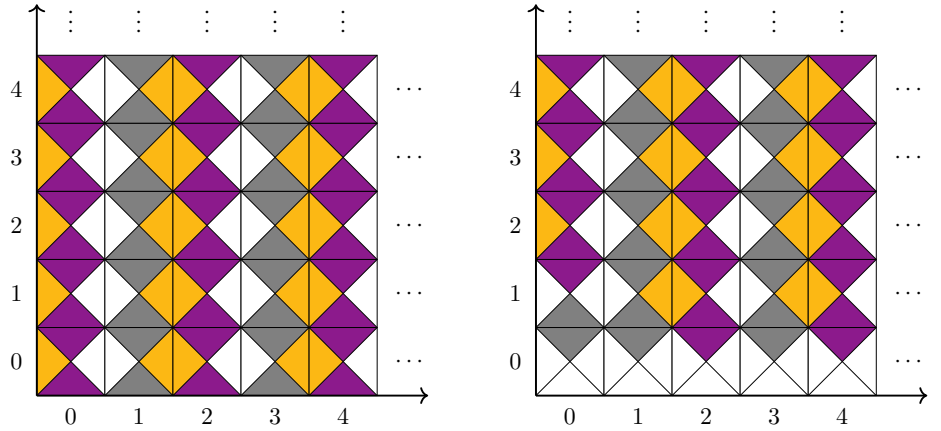


FIGURE 2.3. Deux pavages du plan possibles avec le catalogue C_{ex} de la figure 2.2.

pour le catalogue de la figure 2.2, on a les contraintes suivantes :

$$\begin{aligned}
 H &= \{(t_0, t_0), (t_0, t_1), (t_0, t_2), (t_0, t_3), (t_1, t_0), (t_1, t_1), (t_1, t_2), (t_1, t_3), \\
 &\quad (t_2, t_0), (t_2, t_1), (t_2, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_0), (t_4, t_1), (t_4, t_2), (t_4, t_3)\} \\
 V &= \{(t_0, t_2), (t_0, t_3), (t_1, t_4), (t_2, t_4), (t_3, t_2), (t_3, t_3), (t_4, t_4)\}
 \end{aligned}$$

Un *pavage du plan par C* est alors une fonction $p: \mathbb{N} \times \mathbb{N} \rightarrow C$ telle que

- (1) si $p(i, j) = t$ et $p(i + 1, j) = t'$ alors $(t, t') \in H$ et
- (2) si $p(i, j) = t$ et $p(i, j + 1) = t'$ alors $(t, t') \in V$.

Le problème de décision correspondant est le suivant.

Problème (PAVAGE DU PLAN).

instance : un catalogue C et deux relations $H, V \subseteq C \times C$

question : est-ce qu'il existe un pavage du plan par C ?

Le problème de PAVAGE DU PLAN n'a pas de solution algorithmique.

Théorème 2.14 (BERGER–GUREVICH et KORYAKOV). *Le problème de PAVAGE DU PLAN est indécidable.*

On peut modéliser un problème de pavage comme un problème de satisfaction. On utilise pour cela une relation binaire non interprétée P_t pour chaque tuile $t \in C$ du catalogue; l'idée ici étant que $(x, y) \in P_t$ si $p(x, y) = t$. On écrit alors une formule close

$$\varphi_C \stackrel{\text{def}}{=} \forall x \exists x' \forall y. \varphi_1(x, y) \wedge \varphi_H(x, x', y) \wedge \varphi_V(x, x', y) \quad (2.16)$$

qui force les relations P_t à coder une fonction de pavage p en demandant que chaque position (x, y) soit associée à une unique tuile par

$$\varphi_1(x, y) \stackrel{\text{def}}{=} \bigwedge_{t \neq t'} \neg P_t(x, y) \vee \neg P_{t'}(x, y), \quad (2.17)$$

en vérifiant les contraintes horizontales par

$$\varphi_H(x, x', y) \stackrel{\text{def}}{=} \bigvee_{(t,t') \in H} P_t(x, y) \wedge P_{t'}(x', y), \quad (2.18)$$

et en vérifiant les contraintes verticales par

$$\varphi_V(x, x', y) \stackrel{\text{def}}{=} \bigvee_{(t,t') \in V} P_t(y, x) \wedge P_{t'}(y, x'). \quad (2.19)$$

Proposition 2.15. *La formule φ_C est satisfiable si et seulement si on peut paver le plan avec les tuiles du catalogue C .*

Démonstration. Notons tout d'abord que la skolémisation de φ_C est la formule

$$\varphi'_C \stackrel{\text{def}}{=} \forall x \forall y. \varphi_1(x, y) \wedge \varphi_H(x, f(x), y) \wedge \varphi_V(x, f(x), y)$$

où l'on a introduit un nouveau symbole de fonction unaire f . La formule φ'_C est équi-satisfiable avec φ , donc il suffit de montrer que φ'_C est satisfiable si et seulement si on pouvait paver le plan avec les tuiles du catalogue C .

Si on peut paver le plan par une fonction $p: \mathbb{N} \times \mathbb{N} \rightarrow C$, alors il existe une interprétation I telle que $I \models \varphi'_C$. On définit pour cela le domaine $D_I \stackrel{\text{def}}{=} \mathbb{N}$ où f est interprétée comme la fonction successeur $f^I: n \mapsto n + 1$ et chaque P_t pour $t \in C$ est interprétée par la relation $P_t^I \stackrel{\text{def}}{=} \{(i, j) \mid p(i, j) = t\}$. La formule φ'_C est donc satisfiable dans ce cas.

Inversement, si φ'_C est satisfiable, alors il existe une interprétation I telle que $I \models \varphi'_C$. Par le théorème de HERBRAND, on peut supposer sans perte de généralité que I a pour domaine $D_I \stackrel{\text{def}}{=} T(\mathcal{F})$ l'ensemble des termes clos sur l'ensemble de symboles de fonctions $\mathcal{F} = \{a^{(0)}, f^{(1)}\}$ et interprète a comme le terme constitué d'une feuille étiquetée par a et f comme la fonction $f^I: u \mapsto f(u)$ qui rajoute une nouvelle racine étiquetée par f au-dessus du terme u . On peut alors observer que $I \models \varphi'_C$ implique l'existence d'un pavage du plan $p: \mathbb{N} \times \mathbb{N} \rightarrow C$ où $p(i, j) = t$ pour $(i, j) \in \mathbb{N} \times \mathbb{N}$ si et seulement si la paire de termes

$$\left(\underbrace{f(f(\dots(f(a)\dots))}_{i \text{ fois}}, \underbrace{f(f(\dots(f(a))))}_{j \text{ fois}} \right)$$

appartient à l'interprétation P_t^I de P_t . □

On peut déduire le corollaire suivant du théorème de BERGER–GUREVICH et KORYAKOV.

Corollaire 2.16 (indécidabilité de la SATISFIABILITÉ). *Le problème de SATISFIABILITÉ est indécidable.*

2.5.2. *Le cas des structures finies.* On peut reprendre les questions d'évaluation dans une structure et de satisfiabilité dans le cas des structures finies; ce cas est particulièrement intéressant en lien avec les bases de données, qui comme vu dans l'exemple 2.5 ne sont après tout rien d'autre que des structures finies sur une signature relationnelle.

Les deux problèmes de décision qui nous intéressent ici sont les suivants :

Problème (ÉVALUATION FINIE).

instance : une interprétation I de domaine D_I fini et une formule close φ

question : est-ce que $I \models \varphi$?

▲ La skolémisation et le théorème de HERBRAND ne sont pas au programme.

☞ L'indécidabilité du problème de SATISFIABILITÉ découle aussi de l'indécidabilité de l'Entscheidungsproblem, qui a été démontrée par CHURCH et TURING dans les années 1930.

■ Voir (DUPARC, 2015, thm. 489), (GOUBAULT-LARRECQ et MACKIE, 1997, thm. 6.20), (CORI et LASCAR, 2003, cor. 4.2), (DAVID, NOUR et RAFFALLI, 2003, cor. 3.6.7).

Problème (SATISFIABILITÉ FINIE).**instance** : une formule close φ **question** : existe-t'il une interprétation I de domaine D_I fini telle que $I \models \varphi$?

À noter qu'une interprétation I de domaine fini peut être décrite en donnant explicitement les tables des fonctions f^I et les tuples des relations R^I . Dans ces deux problèmes, on se concentre sur des formules closes pour simplifier la formulation.

■ (LIBKIN, 2004, thm. 6.16)

☞ Cette démonstration montre que l'ÉVALUATION FINIE est PSPACE-difficile même si on fixe I .

☞ Dans le cas des bases de données relationnelles, on s'intéresse plutôt à la complexité en la taille des données : on fixe la taille de la formule φ . On obtient alors une complexité dans la classe uniform-AC₀ (voir LIBKIN, 2004, chap. 6) et une complexité paramétrée AW[*]-complète (voir FLUM et GROHE, 2006, sec. 8.6).

■ (LIBKIN, 2004, thm. 9.2). L'énoncé (et la preuve) du théorème peut être renforcé pour ne nécessiter qu'un seul symbole de relation binaire dans la signature.

Théorème 2.17 (STOCKMEYER). *Le problème d'ÉVALUATION FINIE est PSPACE-complet.*

Démonstration. Commençons par noter que, dans le cas d'une interprétation I de domaine fini, l'évaluation inductive de $\llbracket \varphi \rrbracket_\rho^I$ se fait en espace polynomial en la cardinalité $|D_I|$ du domaine et la taille de φ .

Pour montrer que le problème est PSPACE-difficile, on va exhiber une réduction depuis QBF. Pour rappel, l'entrée de ce problème est une *formule booléenne quantifiée*, c'est-à-dire une formule de la forme $Q_1 P_1 \cdots Q_n P_n \cdot \psi$ où les P_1, \dots, P_n sont des propositions, chaque Q_i est soit \exists soit \forall , et ψ est une formule propositionnelle sur P_1, \dots, P_n , et la question est de savoir si la formule quantifiée s'évalue à **V** ou **F** quand on fait les quantifications des propositions sur \mathbb{B} .

On considère pour la réduction la signature $L \stackrel{\text{def}}{=} (\emptyset, \{V^{(1)}\})$ et l'interprétation $I \stackrel{\text{def}}{=} (\mathbb{B}, \{\mathbf{V}\})$, donc telle que $V^I(\mathbf{V}) = \mathbf{V}$ et $V^I(\mathbf{F}) = \mathbf{F}$.

À partir d'une formule booléenne quantifiée $Q_1 P_1 \cdots Q_n P_n \cdot \psi$, on construit une formule close du premier ordre $Q_1 x_1 \cdots Q_n x_n \cdot \varphi$ où φ est la formule ψ dans laquelle on a remplacé chaque occurrence d'une proposition P_i par la formule atomique $V(x_i)$. On a bien que $Q_1 P_1 \cdots Q_n P_n \cdot \psi$ s'évalue à **V** si et seulement si $I \models Q_1 x_1 \cdots Q_n x_n \cdot \varphi$. \square

Théorème 2.18 (TRAKHTENBROT). *Le problème de SATISFIABILITÉ FINIE est indécidable.*

Démonstration. On exhibe une réduction depuis le problème de l'ARRÊT. Soit $\langle \mathcal{M}, w \rangle$ une instance de ce problème, où $\mathcal{M} = (Q, \Sigma, \Gamma, b, \delta, q_0, q_a, q_r)$ est une machine de TURING déterministe à une bande, où Q dénote son ensemble fini d'états, Σ son alphabet fini d'entrée, Γ son alphabet fini de travail, $b \in \Gamma \setminus \Sigma$ un symbole blanc, $\delta: Q \times \Gamma \rightarrow (Q \setminus \{q_a, q_r\}) \times \Gamma \times \{-1, 0, 1\}$ sa fonction de transition (où $-1, 0$ et 1 dénotent un mouvement vers la gauche, sur place ou vers la droite), $q_0 \in Q$ son état initial, $q_a \in Q$ son état d'acceptation et $q_r \in Q$ son état de rejet, et $w \in \Sigma^*$ est un mot d'entrée. Sans perte de généralité, on peut supposer que w est le mot vide et que la machine ne va jamais à gauche de sa position initiale.

Nous allons construire une formule close $\varphi_{\mathcal{M}}$ qui sera satisfiable dans une interprétation de domaine fini si et seulement si la machine \mathcal{M} s'arrête sur le mot vide. La signature que nous allons utiliser est $L \stackrel{\text{def}}{=} (\{0^{(0)}\}, \{\leq^{(2)}\}, (R_a^{(2)})_{a \in \Gamma}, (T_q^{(2)})_{q \in Q})$.

Domaine ordonné. On va commencer par imposer que, si $I \models \varphi_{\mathcal{M}}$ est de domaine fini, alors \leq^I est un pré-ordre total avec 0^I comme élément minimal :

$$\begin{aligned} \forall x. x \leq x & && \text{(réflexivité)} \\ \forall x \forall y \forall z. x \leq y \wedge y \leq z \rightarrow x \leq z & && \text{(transitivité)} \\ \forall x \forall y. x \leq y \vee y \leq x & && \text{(totalité)} \\ \forall x. 0 \leq x & && \text{(élément minimal)} \end{aligned}$$

On définit la formule « $x = y$ » par $x \leq y \wedge y \leq x$ et on note $=^I \stackrel{\text{def}}{=} \leq^I \cap \geq^I$; alors $=^I$ est une relation d'équivalence sur D_I et on impose que ce soit une congruence pour les autres relations :

$$\bigwedge_{a \in \Gamma} \forall x_1 \forall x_2 \forall y_1 \forall y_2. x_1 = y_1 \wedge x_2 = y_2 \rightarrow (R_a(x_1, x_2) \rightarrow R_a(y_1, y_2)) \quad (R_a\text{-congruence})$$

$$\bigwedge_{q \in Q} \forall x_1 \forall x_2 \forall y_1 \forall y_2. x_1 = y_1 \wedge x_2 = y_2 \rightarrow (T_q(x_1, x_2) \rightarrow T_q(y_1, y_2)) \quad (T_q\text{-congruence})$$

Sans perte de généralité, on pourra dès lors supposer que I est une interprétation *normale*, c'est-à-dire que $=^I$ est l'identité sur I , et donc que \leq^I est un ordre total avec un unique élément minimal 0^I . Comme D_I est fini, cet ordre est discret, et on peut définir la notion de successeur et de prédécesseur via la formule « $x + 1 = y$ » définie comme $x \leq y \wedge \neg(y \leq x) \wedge \forall z. x \leq z \wedge z \leq y \rightarrow z \leq x \vee y \leq z$ et la formule « $x + (-1) = y$ » définie comme $y + 1 = x$; on note aussi « $x + 0 = y$ » pour la formule $x = y$.

Prédicats sur les configurations. Le domaine D_I d'une interprétation I telle que $I \models \varphi_{\mathcal{M}}$ va nous servir à la fois pour représenter des positions p sur la bande de la machine de TURING et des configurations t successives de la machine. L'intuition derrière les symboles R_a est que $R_a^I(p, t)$ soit vrai quand la position p de la bande lors de la t^{e} configuration contient le symbole $a \in \Gamma$, et que $T_q^I(p, t)$ soit vrai quand la machine est dans l'état $q \in Q$ avec sa tête de lecture en position p lors de la t^{e} configuration.

Pour que cette représentation soit correcte, il faut garantir qu'à tout moment, il y ait exactement un symbole de Γ par position sur la bande, un état courant, et une position de la tête :

$$\begin{aligned} & \forall y. \left(\forall x. \bigvee_{a \in \Gamma} (R_a(x, y) \wedge \bigwedge_{c \in \Gamma \setminus \{a\}} \neg R_c(x, y)) \right) && \text{(unique symbole par position)} \\ & \wedge \left(\exists x. \bigvee_{q \in Q} (T_q(x, y) \wedge \bigwedge_{p \in Q \setminus \{q\}} \neg T_p(x, y)) \right) && \text{(unique état)} \\ & \wedge \forall z. \neg(x = z) \rightarrow \bigwedge_{p \in Q} \neg T_p(z, y) && \text{(unique position de tête)} \end{aligned}$$

On impose aussi que la configuration initiale commence avec la tête en position 0^I dans l'état q_0 et que la bande soit initialement blanche :

$$T_{q_0}(0, 0) \wedge \forall x. R_b(x, 0) \quad \text{(configuration initiale)}$$

On impose ensuite que les configurations successives respectent la fonction de transition δ ; si $\delta(q, a) = (p, c, d) \in Q \times \Gamma \times \{-1, 0, 1\}$ on note $\delta_Q(q, a) \stackrel{\text{def}}{=} p$, $\delta_\Gamma(q, a) \stackrel{\text{def}}{=} c$ et

$\delta_{\pm 1}(q, a) \stackrel{\text{def}}{=} d :$

$$\begin{aligned} \forall x \forall y. \bigwedge_{q \in Q, a \in \Gamma} T_q(x, y) \wedge R_a(x, y) \rightarrow \exists x' \exists y'. x + \delta_{\pm 1}(q, a) = x' \wedge y + 1 = y' \\ \wedge T_{\delta_Q(q, a)}(x', y') \quad \text{(mouvement de tête et nouvel état)} \\ \wedge R_{\delta_\Gamma(q, a)}(x, y') \quad \text{(nouveau contenu à la position de tête)} \\ \wedge \forall z. \neg(z = x) \bigwedge_{c \in \Gamma} R_c(z, y') \leftrightarrow R_c(z, y) \quad \text{(recopie du reste de la bande)} \end{aligned}$$

Enfin, on demande à ce que la machine \mathcal{M} s'arrête à un moment en allant dans l'état q_a d'acceptation ou q_r de rejet :

$$\exists x \exists y. T_{q_a}(x, y) \vee T_{q_r}(x, y) \quad \text{(arrêt)}$$

La formule $\varphi_{\mathcal{M}}$ est la conjonction de toutes les contraintes précédentes, et garantit que, s'il existe une interprétation I de domaine fini telle que $I \models \varphi_{\mathcal{M}}$, alors la machine \mathcal{M} s'arrête sur l'entrée vide. Inversement, si la machine s'arrête, alors il existe bien une interprétation de domaine fini qui satisfait toutes les contraintes de $\varphi_{\mathcal{M}}$, construite à partir de l'unique exécution de \mathcal{M} . \square

3. SUBSTITUTIONS

Résumé. Une *substitution* est une fonction σ de domaine fini qui associe à toute variable $x \in X$ un terme $\sigma(x) \in T(\mathcal{F}, X)$; par extension, $t\sigma$ est le terme t dans lequel toutes les occurrences de chaque variable x ont été remplacées par $\sigma(x)$.

Une substitution σ est *applicable* à une formule φ si elle n'interagit pas avec les variables liées de φ ; on note alors « $\varphi\sigma$ » pour la formule φ dans laquelle toutes les occurrences de chaque variable x ont été remplacées par $\sigma(x)$. Modulo *α -renommage*, qui consiste à renommer les variables liées de φ , on peut toujours appliquer une substitution (c.f. propriété 3.3).

Soit I une interprétation et ρ une valuation. On dénote par « $\rho\sigma$ » la valuation qui associe pour toute variable $x \in X$ l'élément $(\rho\sigma)(x) \stackrel{\text{def}}{=} \llbracket \sigma(x) \rrbracket_{\rho}^I$; alors le lemme 3.5 de substitution dit que $\llbracket \varphi\sigma \rrbracket_{\rho}^I = \llbracket \varphi \rrbracket_{\rho\sigma}^I$.

Deux termes t et t' de $T(\mathcal{F}, X)$ s'*unifient* s'il existe une substitution σ telle que $t\sigma = t'\sigma$, appelée leur *unificateur*. On définit le pré-ordre « \lesssim » entre substitutions par $\sigma \lesssim \sigma'$ s'il existe une substitution τ telle que $\sigma' = \sigma\tau$; on dira que σ est plus *générale* que σ' . S'il existe un unificateur de t et t' , alors il en existe un *plus général* (à renommage $\sim \stackrel{\text{def}}{=} \lesssim \cap \gtrsim$ près), noté $\text{mgu}(t \stackrel{?}{=} t')$. Deux algorithmes d'unification sont présentés en sections 3.3 et 3.4, le premier en temps exponentiel et le second en temps presque linéaire.

Une *substitution* est une fonction $\sigma: X \rightarrow T(\mathcal{F}, X)$ de *domaine* $\text{dom}(\sigma) \stackrel{\text{def}}{=} \{x \in X \mid \sigma(x) \neq x\}$ fini. On note aussi σ comme $[\sigma(x_1)/x_1, \dots, \sigma(x_n)/x_n]$ où x_1, \dots, x_n sont les variables distinctes de $\text{dom}(\sigma)$.

Une substitution définit une fonction $t \mapsto t\sigma$ de $T(\mathcal{F}, X)$ dans $T(\mathcal{F}, X)$ par induction sur le terme t

$$x\sigma \stackrel{\text{def}}{=} \sigma(x), \quad f(t_1, \dots, t_m)\sigma \stackrel{\text{def}}{=} f(t_1\sigma, \dots, t_m\sigma)$$

pour tout $m \in \mathbb{N}$ et tout $f \in \mathcal{F}_m$. Une substitution à image dans $T(\mathcal{F})$ est dite *close*.

Ces définitions s'étendent aussi aux formules. Cependant, du fait de la présence de variables liées dans les formules, une substitution n'est pas applicable à n'importe quelle formule. Pour une substitution σ et une formule φ , on note $\text{Images}(\sigma) \stackrel{\text{def}}{=} \bigcup_{x \in \text{dom}(\sigma)} \text{Libres}(\sigma(x))$ son ensemble de variables images (« *range variables* » en anglais). On dit que σ est *applicable* à φ si

$$(\text{dom}(\sigma) \cup \text{Images}(\sigma)) \cap \text{Liées}(\varphi) = \emptyset,$$

■ (DUPARC, 2015, sec. III.10.7),
(GOUBAULT-LARRECQ et
MACKIE, 1997, def. 6.4),
(HARRISSON, 2009, sec. 3.4).

et dans ce cas on définit $\varphi\sigma$ comme l'application de σ à φ par induction sur la formule φ

$$\begin{aligned} R(t_1, \dots, t_m)\sigma &\stackrel{\text{def}}{=} R(t_1\sigma, \dots, t_m\sigma), & (\neg\varphi)\sigma &\stackrel{\text{def}}{=} \neg(\varphi\sigma), \\ (\varphi \vee \psi)\sigma &\stackrel{\text{def}}{=} (\varphi\sigma) \vee (\psi\sigma), & (\exists x.\varphi)\sigma &\stackrel{\text{def}}{=} \exists x.(\varphi\sigma), \end{aligned}$$

où $m \in \mathbb{N}$ et $R \in \mathcal{P}_m$.

La *composition* $\sigma\sigma'$ de deux substitutions σ et σ' est définie par $\varphi(\sigma\sigma') \stackrel{\text{def}}{=} (\varphi\sigma)\sigma'$. Par exemple $(B(x) \vee B(y))[y/x, z/y] = B(y) \vee B(z)$, $(B(x) \vee B(y))[y/x][z/y] = B(z) \vee B(z)$, et $(B(x) \vee B(y))[z/x] = B(z) \vee B(y)$.

3.1. Lemme de substitution. Les valuations fournissent le pendant côté modèles des substitutions côté formules. Pour une substitution σ et une valuation ρ , notons $\sigma\rho$ la valuation $x \mapsto \llbracket \sigma(x) \rrbracket_\rho^I$ pour tout $x \in X$. Nous préparons ici le terrain pour le lemme 3.5 de substitution, qui sera énoncé sous une forme plus générale un peu plus loin.

Lemme 3.1. *Pour tout terme t , toute substitution σ , toute interprétation I , et toute valuation ρ , $\llbracket t\sigma \rrbracket_\rho^I = \llbracket t \rrbracket_{\sigma\rho}^I$.*

Démonstration. Par induction structurelle sur t .

Pour le cas de base d'une variable x , $\llbracket x\sigma \rrbracket_\rho^I = \llbracket \sigma(x) \rrbracket_\rho^I = (\sigma\rho)(x) = \llbracket x \rrbracket_{\sigma\rho}^I$.

Pour un terme $f(t_1, \dots, t_m)$, $\llbracket f(t_1, \dots, t_m)\sigma \rrbracket_\rho^I = \llbracket f(t_1\sigma, \dots, t_m\sigma) \rrbracket_\rho^I = f^I(\llbracket t_1\sigma \rrbracket_\rho^I, \dots, \llbracket t_m\sigma \rrbracket_\rho^I) \stackrel{\text{h.i.}}{=} f^I(\llbracket t_1 \rrbracket_{\sigma\rho}^I, \dots, \llbracket t_m \rrbracket_{\sigma\rho}^I) = \llbracket f(t_1, \dots, t_m) \rrbracket_{\sigma\rho}^I$. \square

Lemme 3.2. *Pour toute formule φ , toute substitution σ applicable à φ , toute interprétation I , et toute valuation ρ , $\llbracket \varphi\sigma \rrbracket_\rho^I = \llbracket \varphi \rrbracket_{\sigma\rho}^I$.*

Démonstration. Par induction structurelle sur φ .

Pour une formule atomique $R(t_1, \dots, t_m)$, une négation $\neg\varphi$, ou une disjonction $\varphi \vee \psi$, cela découle du lemme 3.1 et de l'hypothèse d'induction. Pour une quantification $\exists x.\varphi$,

$$\llbracket (\exists x.\varphi)\sigma \rrbracket_\rho^I = \llbracket \exists x.(\varphi\sigma) \rrbracket_\rho^I = \text{Ou}_{e \in D_I} \llbracket \varphi\sigma \rrbracket_{\rho[e/x]}^I \stackrel{\text{h.i.}}{=} \text{Ou}_{e \in D_I} \llbracket \varphi \rrbracket_{\sigma(\rho[e/x])}^I = \llbracket \exists x.\varphi \rrbracket_{\sigma\rho}^I,$$

où il faut montrer pour justifier cette dernière étape que, pour toute variable libre $z \in \text{Libres}(\varphi)$, $(\sigma(\rho[e/x]))(z) = ((\sigma\rho)[e/x])(z)$, ce qui permettra de conclure par la propriété 2.6. Si $z = x$, alors $(\sigma(\rho[e/x]))(x) = \llbracket \sigma(x) \rrbracket_{\rho[e/x]}^I = \llbracket x \rrbracket_{\rho[e/x]}^I = e = \llbracket x \rrbracket_{(\sigma\rho)[e/x]}^I = ((\sigma\rho)[e/x])(x)$ où $\sigma(x) = x$ puisque, comme σ est applicable à $\exists x.\varphi$ et $x \in \text{Liées}(\exists x.\varphi)$, $x \notin \text{dom}(\sigma)$. Si $z \neq x$, alors $(\sigma(\rho[e/x]))(z) = \llbracket \sigma(z) \rrbracket_{\rho[e/x]}^I = \llbracket \sigma(z) \rrbracket_\rho^I = ((\sigma\rho)[e/x])(z)$, où l'égalité centrale repose sur le fait que $x \notin \text{Libres}(\sigma(z))$, qui à son tour découle de $\text{Liées}(\exists x.\varphi) \cap \text{Images}(\sigma) = \emptyset$ puisque σ est applicable à $\exists x.\varphi$. \square

☞ L' α -renommage est aussi une notion de base du λ -calcul, l'archétype de langage de programmation fonctionnelle qui donne son nom (entre autres) aux lambda-expressions en OCaml.

3.2. α -renommages. Quand une substitution σ n'est pas applicable à une formule φ , il est cependant possible d'effectuer un α -renommage à φ pour obtenir une formule φ' sur laquelle appliquer σ . On raisonnera donc implicitement non sur des formules mais sur des classes de formules équivalentes à α -renommage près. Il est important dans ce cas que cette opération définisse une congruence (l' α -congruence) sur les formules pour pouvoir continuer à raisonner inductivement, et que l'opération préserve la sémantique des formules.

On définit l'opération d' α -renommage comme une règle de réécriture sur les formules

$$\exists x.\varphi \rightarrow_{\alpha} \exists y.\varphi[y/x] \quad (\alpha\text{-renommage})$$

où $y \notin \text{Libres}(\exists x.\varphi)$ et $[y/x]$ est applicable à φ . À noter que l' α -renommage n'impacte que les variables liées de φ , ce qui explique pourquoi il en préserve la sémantique (voir le lemme 3.4 ci-dessous).

On définit l' α -congruence $=_{\alpha}$ comme la plus petite congruence entre formules engendrée par \rightarrow_{α} , c'est-à-dire la plus petite relation réflexive, symétrique et transitive telle que $\varphi \rightarrow_{\alpha} \psi$ implique $\varphi =_{\alpha} \psi$ et telle que si $\varphi =_{\alpha} \psi$ et $\varphi' =_{\alpha} \psi'$, alors $\neg\varphi =_{\alpha} \neg\psi$, $\varphi \vee \varphi' =_{\alpha} \psi \vee \psi'$ et $\exists x.\varphi =_{\alpha} \exists x.\psi$.

En appliquant des α -renommages inductivement sur les sous-formules croissantes de φ , on peut au besoin renommer toutes les variables liées de φ , et on déduit :

Propriété 3.3 (applicabilité). *Pour toute substitution σ et toute formule φ , il existe une formule φ' telle que $\varphi =_{\alpha} \varphi'$ et que σ soit applicable à φ' . De plus, si $\varphi =_{\alpha} \varphi''$ et σ est aussi applicable à φ'' , alors $\varphi' =_{\alpha} \varphi''$ et $\varphi'\sigma =_{\alpha} \varphi''\sigma$.*

Il reste à vérifier que les α -renommages n'impactent pas la sémantique des formules.

Lemme 3.4 (α -congruence). *Soient φ et ψ deux formules telles que $\varphi =_{\alpha} \psi$. Alors pour toute interprétation I et toute valuation ρ , $\llbracket \varphi \rrbracket_{\rho}^I = \llbracket \psi \rrbracket_{\rho}^I$.*

Démonstration. Par induction sur la congruence $=_{\alpha}$.

Le cas de base est celui d'un α -renommage $\exists x.\varphi \rightarrow_{\alpha} \exists y.\varphi[y/x]$ avec $y \notin \text{Libres}(\exists x.\varphi)$ et $[y/x]$ applicable. Puisque $[y/x]$ est applicable, par le lemme 3.2,

$$\llbracket \exists y.\varphi[y/x] \rrbracket_{\rho}^I = \text{Ou}_{e \in D_I} \llbracket \varphi[y/x] \rrbracket_{\rho[e/y]}^I = \text{Ou}_{e \in D_i} \llbracket \varphi \rrbracket_{[y/x](\rho[e/y])}^I = \text{Ou}_{e \in D_i} \llbracket \varphi \rrbracket_{\rho[e/x]}^I = \llbracket \exists x.\varphi \rrbracket_{\rho}^I$$

où nous devons justifier pour l'avant-dernière étape que, pour toute variable libre $z \in \text{Libres}(\varphi)$, $([y/x](\rho[e/y]))(z) = (\rho[e/x])(z)$, ce qui permettra de conclure ce cas de base par la propriété 2.6. Si $x = z$, alors $([y/x](\rho[e/y]))(x) = \llbracket y \rrbracket_{\rho[e/y]}^I = e = \llbracket x \rrbracket_{\rho[e/x]}^I = (\rho[e/x])(x)$. Si $x \neq z$, alors d'une part $([y/x](\rho[e/y]))(z) = \llbracket z \rrbracket_{\rho[e/y]}^I = \llbracket z \rrbracket_{\rho}^I$ puisque $y \notin \text{Libres}(\exists x.\varphi)$ et donc $y \neq z$, et d'autre part $(\rho[e/x])(z) = \llbracket z \rrbracket_{\rho[e/x]}^I = \llbracket z \rrbracket_{\rho}^I$.

Le cas de base de la réflexivité ainsi que les étapes d'induction pour la symétrie, la transitivité, et la congruence pour \neg , \vee et \exists sont évidents puisque $\llbracket \varphi \rrbracket_{\rho}^I = \llbracket \psi \rrbracket_{\rho}^I$ vue comme une relation entre formules est aussi une congruence. \square

La propriété 3.3 et le lemme 3.4 justifient un abus de notation : nous écrirons désormais « $\varphi\sigma$ » pour une formule $\varphi'\sigma$ où $\varphi =_{\alpha} \varphi'$ et σ est applicable à φ' . Le lemme 3.1 ainsi que le lemme 3.4 d' α -congruence combiné au lemme 3.2 nous permettent alors d'énoncer le lemme de substitution.

Lemme 3.5 (substitution). *Pour tout terme t , toute formule φ , toute substitution σ , toute interprétation I , et toute valuation ρ , $\llbracket t\sigma \rrbracket_{\rho}^I = \llbracket t \rrbracket_{\sigma\rho}^I$ et $\llbracket \varphi\sigma \rrbracket_{\rho}^I = \llbracket \varphi \rrbracket_{\sigma\rho}^I$.*

■ (GOUBAULT-LARRECQ et MACKIE, 1997, thm. 6.8).

On utilisera par la suite deux identités aisément démontrables par induction sur φ : en appliquant au besoin une α -congruence à φ pour rendre les substitutions applicables, si $x \notin \text{Libres}(\varphi)$, alors

$$\varphi[t/x] =_{\alpha} \varphi, \quad (3.1)$$

et si $x \notin \text{Libres}(\varphi) \setminus \{y\}$, alors

$$\varphi[x/y][t/x] =_{\alpha} \varphi[t/y]. \quad (3.2)$$

Exemple 3.6 (propriétés arithmétiques). Dans la signature arithmétique avec $\mathcal{F} \stackrel{\text{def}}{=} \{+(^{(2)}, \times^{(2)})\}$ et $\mathcal{P} \stackrel{\text{def}}{=} \{=(^{(2)})\}$ de l'exemple 2.3 et l'interprétation $I = (\mathbb{N}, +, \times)$, on peut définir dans la logique :

- le nombre 0 par une formule avec une variable libre

$$\text{zero}(x) \stackrel{\text{def}}{=} x + x = x. \quad (3.3)$$

Pour rappel, si « φ » est le nom d'une formule qui n'est pas close, on peut expliciter que $\text{Libres}(\varphi) = \{x_1, \dots, x_n\}$ en écrivant « $\varphi(x_1, \dots, x_n)$ ». C'est ce que l'on a fait ici avec la formule « *zero* ». De manière très naturelle, on écrit alors « *zero*(y) » pour le résultat $\text{zero}[y/x]$ du renommage de x en y dans la formule *zero*.

Pour revenir à la sémantique de notre formule (3.3), on a $I \models \exists x.\text{zero}(x) \wedge \forall y.(\text{zero}(y) \rightarrow x = y)$, c'est-à-dire qu'il y a exactement un élément e du domaine $D_I \stackrel{\text{def}}{=} \mathbb{N}$ tel que $I, \rho[e/x] \models \text{zero}(x)$: c'est $e = 0$.

- de même, le nombre 1 par une formule avec une variable libre

$$\text{un}(x) \stackrel{\text{def}}{=} \neg \text{zero}(x) \wedge x \times x = x \quad (3.4)$$

et alors $I \models \exists x.\text{un}(x) \wedge \forall y.(\text{un}(y) \rightarrow x = y)$;

- l'ordre strict par une formule avec deux variables libres

$$x < y \stackrel{\text{def}}{=} \exists z. \neg \text{zero}(z) \wedge y = x + z; \quad (3.5)$$

- le prédicat

$$\text{pair}(x) \stackrel{\text{def}}{=} \exists y \exists z. x = y \times (z + z) \wedge \text{un}(z); \quad (3.6)$$

- le prédicat

$$\text{premier}(x) \stackrel{\text{def}}{=} \neg \text{zero}(x) \wedge \neg \exists y \exists z. x = y \times z \wedge \neg \text{un}(y) \wedge \neg \text{un}(z); \quad (3.7)$$

- une formule close qui dit que tout nombre premier supérieur à 2 est impair :

$$\forall x.(\text{premier}(x) \wedge \exists y. x > y + y \wedge \text{un}(y)) \rightarrow \neg \text{pair}(x). \quad (3.8)$$

■ La référence principale pour cette sous-section et la suivante est (BAADER et NIPKOW, 1998). L'unification est utilisée entre autres au sein des systèmes de preuve par résolution pour la logique du premier ordre et pour l'inférence de type dans les langages fonctionnels.

3.3. Unification. Deux termes t et t' de $T(\mathcal{F}, X)$ s'unifient s'il existe une substitution σ telle que $t\sigma = t'\sigma$, appelée leur *unificateur*. On définit le pré-ordre « \lesssim » entre substitutions par $\sigma \lesssim \sigma'$ s'il existe une substitution τ telle que $\sigma' = \sigma\tau$; on dira que σ est plus *générale* que σ' . On peut montrer que s'il existe un unificateur de t et t' , alors il en existe un *plus général* (à renommage $\sim \stackrel{\text{def}}{=} \lesssim \cap \gtrsim$ près), noté $\text{mgu}(t \stackrel{?}{=} t')$. Plus généralement, on pourra considérer l'unification d'un ensemble fini E d'équations $\{t_1 \stackrel{?}{=} t'_1, \dots, t_n \stackrel{?}{=} t'_n\}$ (existe-t'il σ telle que $t_i\sigma = t'_i\sigma$ pour tout $1 \leq i \leq n$?) ou d'un ensemble T de termes (existe-t'il σ telle que $t\sigma = t'\sigma$ pour tous $t, t' \in T$?), dont on notera les unificateurs les plus généraux $\text{mgu}(E)$ et $\text{mgu}(T)$ respectivement.

Le problème d'unification consiste à déterminer si deux termes donnés sont unifiables, et le cas échéant à retourner leur mgu . L'algorithme « naïf » pour ce problème est adapté de l'algorithme original de ROBINSON et sa présentation dans ces notes est due à MARTELLI et MONTANARI. Cet algorithme est basé sur le système de réécriture de la figure 3.1 qui agit sur des ensembles finis E d'équations. Dans ce système de réécriture, les équations $t \stackrel{?}{=} t'$ sont

■ (BAADER et NIPKOW, 1998, sec. 4.6), (LALEMENT, 1990, sec. 6.2), (DAVID, NOUR et RAFFALLI, 2003, sec 7.2), (GOUBAULT-LARRECQ et MACKIE, 1997, sec. 7.2), (JAUME et al., 2020, sec. 8.3)

vues comme commutatives. Pour un ensemble d'équations $E = \{t_1 \stackrel{?}{=} t'_1, \dots, t_n \stackrel{?}{=} t'_n\}$, on note $\text{Libres}(E) \stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq n} \text{Libres}(t_i) \cup \text{Libres}(t'_i)$, et si σ est une substitution, on note $E\sigma \stackrel{\text{def}}{=} \{t_1\sigma \stackrel{?}{=} t'_1\sigma, \dots, t_n\sigma \stackrel{?}{=} t'_n\sigma\}$. On dénote par $\mathcal{U}(E)$ l'ensemble des unificateurs de E : $\mathcal{U}(\{t_1 \stackrel{?}{=} t'_1, \dots, t_n \stackrel{?}{=} t'_n\}) \stackrel{\text{def}}{=} \{\sigma \mid \forall 1 \leq i \leq n. t_i\sigma = t'_i\sigma\}$.

$$\begin{aligned}
E \cup \{t \stackrel{?}{=} t\} &\rightarrow_u E && \text{(eff)} \\
E \cup \{f(t_1, \dots, t_n) \stackrel{?}{=} f(t'_1, \dots, t'_n)\} &\rightarrow_u E \cup \{t_1 \stackrel{?}{=} t'_1, \dots, t_n \stackrel{?}{=} t'_n\} && \text{(dec)} \\
E \cup \{x \stackrel{?}{=} t\} &\rightarrow_u E[t/x] \cup \{x \stackrel{?}{=} t\} && \text{(elim)} \\
&&& \text{si } x \in \text{Libres}(E), x \notin \text{Libres}(t) \text{ et } (t \notin X \text{ ou } t \in \text{Libres}(E))
\end{aligned}$$

FIGURE 3.1. Les règles de l'algorithme d'unification naïve.

On dit qu'une variable x est *résolue* dans E si $E = \{x \stackrel{?}{=} t\} \cup E'$ et $x \notin (\text{Libres}(t) \cup \text{Libres}(E'))$. L'objectif de l'algorithme d'unification naïve va être de mettre toutes les variables sous forme résolue : un ensemble d'équations est en *forme résolue* s'il est de la forme $\{x_1 \stackrel{?}{=} t_1, \dots, x_n \stackrel{?}{=} t_n\}$ où les variables x_1, \dots, x_n sont résolues.

Lemme 3.7. Si $E = \{x_1 \stackrel{?}{=} t_1, \dots, x_n \stackrel{?}{=} t_n\}$ où x_1, \dots, x_n sont résolues, alors pour tout $\sigma \in \mathcal{U}(E)$, $\sigma = [t_1/x_1, \dots, t_n/x_n]\sigma$.

■ (BAADER et NIPKOW, 1998, lem. 4.6.2).

Démonstration. Soit $\sigma \in \mathcal{U}(E)$. On montre pour cela que, pour toute variable $x \in X$, $x[t_1/x_1, \dots, t_n/x_n]\sigma = x\sigma$. Si $x = x_i$ pour un certain $1 \leq i \leq n$, alors d'une part $x_i[t_1/x_1, \dots, t_n/x_n]\sigma = t_i\sigma$, et d'autre part $x_i\sigma = t_i\sigma$ puisque $\sigma \in \mathcal{U}(E)$. Sinon, on a $x \notin \{x_1, \dots, x_n\}$, et alors $x[t_1/x_1, \dots, t_n/x_n]\sigma = x\sigma$. □

Corollaire 3.8 (formes résolues). Si $E = \{x_1 \stackrel{?}{=} t_1, \dots, x_n \stackrel{?}{=} t_n\}$ où x_1, \dots, x_n sont résolues, alors $\text{mgu}(E) = [t_1/x_1, \dots, t_n/x_n]$.

■ (BAADER et NIPKOW, 1998, lem. 4.6.3).

Démonstration. Posons $\sigma_E \stackrel{\text{def}}{=} [t_1/x_1, \dots, t_n/x_n]$. Alors σ_E est un unificateur : pour tout $1 \leq i \leq n$, $x_i\sigma_E = t_i = t_i\sigma_E$, où la dernière égalité est justifiée par le fait que $\text{dom}(\sigma_E) \cap \text{Libres}(\{t_1, \dots, t_n\}) = \emptyset$. Soit maintenant un unificateur $\sigma \in \mathcal{U}(E)$. Par le lemme 3.7, $\sigma_E\sigma = \sigma$ donc $\sigma_E \preceq \sigma$. On a bien $\sigma_E = \text{mgu}(E)$. □

Proposition 3.9 (terminaison). Le système de la figure 3.1 termine : il n'y a pas de séquence infinie $E_0 \rightarrow_u E_1 \rightarrow_u E_2 \rightarrow_u \dots$.

■ (BAADER et NIPKOW, 1998, lem. 4.6.5).

Démonstration. On définit une fonction de rang r des ensembles finis d'équations dans \mathbb{N}^2 telle que, si $E \rightarrow_u E'$ alors $r(E) >_{\text{lex}} r(E')$ où $<_{\text{lex}}$ est l'ordre lexicographique sur \mathbb{N}^2 . Alors l'existence d'une séquence infinie $E_0 \rightarrow_u E_1 \rightarrow_u E_2 \rightarrow_u \dots$ impliquerait celle d'une descente infinie $r(E_0) >_{\text{lex}} r(E_1) >_{\text{lex}} r(E_2) >_{\text{lex}} \dots$ dans \mathbb{N}^2 , en contradiction avec le fait que $<_{\text{lex}}$ est bien fondé.

On considère pour cela l'ensemble $\text{nrv}(E) \stackrel{\text{def}}{=} \{x \in \text{Libres}(E) \mid x \text{ non résolue}\}$ des variables non résolues de E et la taille $|E| \stackrel{\text{def}}{=} \sum_{t \stackrel{?}{=} t' \in E} |t| + |t'|$ de E . Posons $r(E) \stackrel{\text{def}}{=} (|\text{nrv}(E)|, |E|)$. Il reste à montrer que r est une fonction de rang, c'est-à-dire que $E \rightarrow_u E'$ implique $r(E) >_{\text{lex}} r(E')$. On procède pour cela à une distinction de cas.

Cas de (eff) : alors $\text{nrv}(E') \subseteq \text{nrv}(E)$ et $|E'| = |E| - 2|t|$ et donc $r(E') <_{\text{lex}} r(E)$.

Cas de (dec) : alors $\text{nrv}(E') \subseteq \text{nrv}(E)$ et $|E'| = |E| - 2$ et donc $r(E') <_{\text{lex}} r(E)$.

Cas de (elim) : alors $E = F \cup \{x \stackrel{?}{=} t\}$ et $E' = F[t/x] \cup \{x \stackrel{?}{=} t\}$ où $x \in \text{Libres}(F) \setminus \text{Libres}(t)$ et $t \notin X$ ou $t \in \text{Libres}(F)$. Montrons que $\text{nrv}(E') \subsetneq \text{nrv}(E)$ et donc que $r(E') <_{\text{lex}} r(E)$.

D'une part $x \in \text{Libres}(F)$ donc $x \in \text{nrv}(E)$, et d'autre part $x \notin \text{Libres}(F[t/x])$ et $x \notin \text{Libres}(t)$ donc $x \notin \text{nrv}(E')$.

Considérons maintenant une variable $y \in \text{nrv}(E')$, donc telle que $x \neq y$, et montrons que $y \in \text{nrv}(E)$. Par l'absurde, supposons que $y \in \text{nrv}(E')$ mais que $y \notin \text{nrv}(E)$, c'est-à-dire que $F = F' \cup \{y \stackrel{?}{=} t'\}$ où $y \notin \text{Libres}(t') \cup \text{Libres}(F') \cup \text{Libres}(t)$. On a alors $F[t/x] = F'[t/x] \cup \{y \stackrel{?}{=} t'[t/x]\}$ puisque $x \neq y$. Comme $x \neq y$ et $y \notin \text{Libres}(t') \cup \text{Libres}(F') \cup \text{Libres}(t)$, $y \notin \text{Libres}(t'[t/x]) \cup \text{Libres}(F'[t/x]) \cup \text{Libres}(t) \cup \{x\} = \text{Libres}(t'[t/x]) \cup \text{Libres}(F'[t/x] \cup \{x \stackrel{?}{=} t\})$ donc y est résolue dans E' , contradiction. \square

■ (BAADER et NIPKOW, 1998, lem. 4.6.6).

Proposition 3.10 (correction). *Les règles de la figure 3.1 sont correctes : si $E \rightarrow_u E'$, alors $\mathcal{U}(E) = \mathcal{U}(E')$.*

Démonstration. On procède par analyse de cas, selon la règle de la figure 3.1 employée.

Cas de (eff) : alors $E = E' \cup \{t \stackrel{?}{=} t\}$. On a $\sigma \in \mathcal{U}(E)$ si et seulement si $t\sigma = t\sigma$ et $\sigma \in \mathcal{U}(E')$, ce qui est si et seulement si $\sigma \in \mathcal{U}(E')$.

Cas de (dec) : alors $E = F \cup \{f(t_1, \dots, t_n) \stackrel{?}{=} f(t'_1, \dots, t'_n)\}$ et $E' = F \cup \{t_1 \stackrel{?}{=} t'_1, \dots, t_n \stackrel{?}{=} t'_n\}$. On a les équivalences

$$\begin{aligned} \sigma \in \mathcal{U}(E) &\text{ ssi } f(t_1, \dots, t_n)\sigma = f(t'_1, \dots, t'_n)\sigma \text{ et } \sigma \in \mathcal{U}(F) \\ &\text{ ssi } t_1\sigma = t'_1\sigma, \dots, t_n\sigma = t'_n\sigma \text{ et } \sigma[t/x] \in \mathcal{U}(F) \\ &\text{ ssi } \sigma \in \mathcal{U}(E') . \end{aligned}$$

Cas de (elim) : alors $E = F \cup \{x \stackrel{?}{=} t\}$ et $E' = F[t/x] \cup \{x \stackrel{?}{=} t\}$ avec $x \in \text{Libres}(F) \setminus \text{Libres}(t)$ et $t \notin X$ ou $t \in \text{Libres}(F)$. Comme $x \notin \text{Libres}(t)$, $\{x \stackrel{?}{=} t\}$ est un ensemble d'équations sous forme résolue. Donc par le lemme 3.7, si $x\sigma = t\sigma$ alors $\sigma = [t/x]\sigma$. On a alors les équivalences

$$\begin{aligned} \sigma \in \mathcal{U}(E) &\text{ ssi } x\sigma = t\sigma \text{ et } \sigma \in \mathcal{U}(F) \\ &\text{ ssi } x\sigma = t\sigma \text{ et } [t/x]\sigma \in \mathcal{U}(F) \\ &\text{ ssi } x\sigma = t\sigma \text{ et } \sigma \in \mathcal{U}(F[t/x]) \\ &\text{ ssi } \sigma \in \mathcal{U}(E') . \end{aligned} \quad \square$$

■ (BAADER et NIPKOW, 1998, lem. 4.6.10).

Théorème 3.11 (ROBINSON). *Soit E un ensemble fini d'équations. On peut décider si E est unifiable, et le cas échéant, calculer $\text{mgu}(E)$.*

Démonstration. Étant donné E , on applique les règles de la figure 3.1 dans un ordre quelconque. Par la proposition 3.9, on obtient après un nombre fini d'étapes un système d'équations E' tel que $E \rightarrow_u^* E'$ et aucune règle ne s'applique à E' . Par correction, $\mathcal{U}(E) = \mathcal{U}(E')$. On distingue tout d'abord deux cas particuliers.

- (1) Si $(t \stackrel{?}{=} t') \in E'$ avec $t, t' \notin X$, alors comme (dec) ne s'applique pas, $t = f(t_1, \dots, t_n)$ et $t' = g(t'_1, \dots, t'_m)$ avec $f \neq g$, mais alors $\mathcal{U}(E') = \mathcal{U}(E) = \emptyset$.

(2) Si $(x \stackrel{?}{=} t) \in E'$ avec $x \in \text{Libres}(t)$, alors $\mathcal{U}(E') = \mathcal{U}(E) = \emptyset$.

Supposons maintenant qu'aucun des deux cas précédents ne s'applique et montrons que E' est nécessairement en forme résolue. Observons que E' ne contient donc que des équations de la forme $x \stackrel{?}{=} t$ pour $x \in X$ et $x \notin \text{Libres}(t)$. Soit $x \stackrel{?}{=} t$ une telle équation; écrivons E' comme $E' = \{x \stackrel{?}{=} t\} \uplus E''$. Comme (elim) ne s'applique pas et $x \notin \text{Libres}(t)$, seuls deux cas sont possibles :

- si $x \notin \text{Libres}(E'')$, alors x est résolue;
- sinon t est une variable $y \notin \text{Libres}(E'')$ et comme $x \notin \text{Libres}(y)$, $y \notin \text{Libres}(x)$: alors y est résolue.

Ce raisonnement s'applique à toutes les équations de E' , qui est donc en forme résolue. Or, par le corollaire 3.8, E' est alors unifiable et on peut calculer $\text{mgu}(E')$; comme $\mathcal{U}(E) = \mathcal{U}(E')$ par la proposition 3.10, on a finalement $\text{mgu}(E') = \text{mgu}(E)$. \square

Exemple 3.12. Soit le problème d'unification $f(x, y) \stackrel{?}{=} f(g(y), g(z))$. On a la réécriture suivante dans le système de la figure 3.1 :

$$\{f(x, y) \stackrel{?}{=} f(g(y), g(z))\} \xrightarrow{(\text{dec})} \{x \stackrel{?}{=} g(y), y \stackrel{?}{=} g(z)\} \xrightarrow{(\text{elim})} \{x \stackrel{?}{=} g(g(z)), y \stackrel{?}{=} g(z)\}$$

qui est une forme résolue d'où l'on extrait $\text{mgu}(f(x, y) \stackrel{?}{=} f(g(y), g(z))) = [g(g(y))/x, g(z)/y]$.

Exemple 3.13. Soit le problème d'unification $f(x, y) \stackrel{?}{=} f(g(y), g(x))$. On a la réécriture suivante dans le système de la figure 3.1 :

$$\{f(x, y) \stackrel{?}{=} f(g(y), g(x))\} \xrightarrow{(\text{dec})} \{x \stackrel{?}{=} g(y), y \stackrel{?}{=} g(x)\} \xrightarrow{(\text{elim})} \{x \stackrel{?}{=} g(y), y \stackrel{?}{=} g(g(y))\}.$$

Aucune règle ne s'applique à ce dernier système d'équations qui n'est pas en forme résolue, et en effet les termes n'étaient pas unifiables.

Exemple 3.14. Soit le problème d'unification $\{x_1 \stackrel{?}{=} f(x_2, x_2), x_2 \stackrel{?}{=} f(x_3, x_3), x_3 \stackrel{?}{=} f(x_4, x_4)\}$. On a la réécriture suivante dans le système de la figure 3.1 :

$$\begin{aligned} & \{x_1 \stackrel{?}{=} f(x_2, x_2), x_2 \stackrel{?}{=} f(x_3, x_3), x_3 \stackrel{?}{=} f(x_4, x_4)\} \\ & \xrightarrow{(\text{elim})} \{x_1 \stackrel{?}{=} f(f(x_3, x_3), f(x_3, x_3)), x_2 \stackrel{?}{=} f(x_3, x_3), x_3 \stackrel{?}{=} f(x_4, x_4)\} \\ & \xrightarrow{(\text{elim})} \{x_1 \stackrel{?}{=} f(f(f(x_4, x_4), f(x_4, x_4)), f(f(x_4, x_4), f(x_4, x_4))), \\ & \quad x_2 \stackrel{?}{=} f(f(x_4, x_4), f(x_4, x_4)), x_3 \stackrel{?}{=} f(x_4, x_4)\} \end{aligned}$$

qui est une forme résolue d'où l'on pourrait extraire le mgu. En généralisant cet exemple sur n variables, on peut observer que l'algorithme naïf peut avoir un coût exponentiel, qui est inévitable si l'on souhaite explicitement écrire le mgu.

■ (BAADER et NIPKOW, 1998, ex. 4.6.11).

3.4. * Algorithme d'unification par union-find. L'algorithme naïf peut nécessiter un temps exponentiel. Le problème d'unification peut en fait être résolu en temps déterministe linéaire par un algorithme dû à PATERSON et WEGMAN (1978), mais celui-ci est assez complexe. Nous allons voir un algorithme plus simple dû à HUET (1976, sec. 5.7), basé sur union-find, qui travaille en temps déterministe presque linéaire.

💬 Les structures d'union-find sont appelées « unir & trouver » dans le programme MPI en section 3.3, et peuvent être abordées dans la leçon 4 « Exemples de structures de données. Applications. »

3.4.1. *Treillis des substitutions.* Nous allons voir que l'ensemble des substitutions sur $T(\mathcal{F}, X)$ ordonné par \preceq (une fois quotienté par \sim) forme un treillis complet. Nous allons construire pour cela une correspondance avec un sous-treillis du treillis des relations d'équivalence sur $T(\mathcal{F}, X)$.

On dénote par \sqsubset l'ordre sous-terme strict sur $T(\mathcal{F}, X)$, c'est-à-dire la clôture transitive de la relation $\{(f(t_1, \dots, t_m), t_i) \mid m \in \mathbb{N}, 1 \leq i \leq m, f \in \mathcal{F}_m\}$. Appelons une relation d'équivalence sur $T(\mathcal{F}, X)$ *valide* si elle satisfait les trois propriétés qui suivent :

finitaire : il existe un nombre fini de classes d'équivalence $[x]_{\equiv}$ pour $x \in X$ qui ne soient pas des singletons $\{x\}$,

homogène : $f(t_1, \dots, t_m) \equiv g(t'_1, \dots, t'_n)$ implique $f = g$ (et donc $m = n$) et $t_i \equiv t'_i$ pour tous $1 \leq i \leq m$, et

acyclique : la composition $\sqsubset \circ \equiv$ est bien fondée,

où l'on définit la composition entre relations binaires R et R' par $R \circ R' \stackrel{\text{def}}{=} \{(x, z) \mid \exists y. (x, y) \in R \text{ et } (y, z) \in R'\}$. À noter qu'une équivalence valide est nécessairement une congruence.

Propriété 3.15. *L'ensemble des relations d'équivalence valides sur $T(\mathcal{F}, X)$ ordonné par inclusion forme un treillis complet.*

Démonstration. Il suffit d'observer que si $(\equiv_j)_{j \in J}$ est un ensemble de relations d'équivalence valides sur $T(\mathcal{F}, X)$, alors son intersection $\bigcap_{j \in J} \equiv_j$ est valide. \square

Des substitutions aux équivalences valides. Soit σ une substitution. On définit la relation d'équivalence \equiv_σ sur $T(\mathcal{F}, X)$ par $t \equiv_\sigma t'$ si $t\sigma = t'\sigma$.

Propriété 3.16. *L'équivalence \equiv_σ est valide.*

Démonstration. La relation \equiv_σ est manifestement homogène; elle est finitaire puisque les seules classes $[x]_{\equiv_\sigma} \neq \{x\}$ apparaissent pour $x \neq x\sigma$, c'est-à-dire pour $x \in \text{dom}(\sigma)$, qui est fini. Enfin, $\sqsubset \circ \equiv_\sigma$ est bien fondée, sans quoi on aurait $t_0\sigma = s_0\sigma \sqsubset t_1\sigma = s_1\sigma \sqsubset \dots$ une descente infinie dans $t_0\sigma$ par l'ordre de sous-terme strict. \square

Propriété 3.17. *Si $\sigma \preceq \sigma'$ alors $\equiv_\sigma \subseteq \equiv_{\sigma'}$.*

Démonstration. Soit τ une substitution telle que $\sigma' = \sigma\tau$. Alors pour tous termes t et t' , $t\sigma = t'\sigma$ implique $t\sigma\tau = t'\sigma\tau$. \square

Des équivalences valides aux substitutions. Soit \equiv une relation d'équivalence valide. Pour toute classe d'équivalence e de \equiv , on choisit un représentant canonique $c(e)$ tel que, s'il existe un terme $t \in e \setminus X$ qui ne soit pas une variable, alors $c(e) \notin X$ ne soit pas une variable non plus. Il faut noter ici que le seul véritable choix est celui d'un représentant dans X quand $e \subseteq X$ n'est pas réduit à un singleton : l'homogénéité de \equiv fait que le choix entre différents représentants canoniques quand $e \setminus X \neq \emptyset$ est immatériel.

On ordonne X par $x \prec y$ si $y \sqsubset c([x]_{\equiv})$; comme \equiv est acyclique, \prec est bien fondé. On peut alors définir une fonction $\sigma_{c, \equiv} : X \rightarrow T(\mathcal{F}, X)$ par induction bien fondée sur \prec :

$$\sigma_{c, \equiv}(x) \stackrel{\text{def}}{=} (c([x]_{\equiv}))\sigma_{c, \equiv} .$$

Propriété 3.18. *La fonction $\sigma_{c, \equiv}$ est une substitution telle que, pour tous termes $t, t' \in T(\mathcal{F}, X)$, $t \equiv t'$ implique $t\sigma_{c, \equiv} = t'\sigma_{c, \equiv}$.*

Démonstration. La fonction $\sigma_{c,\equiv}$ est bien une substitution : $\text{dom}(\sigma)_{c,\equiv}$ est fini puisque \equiv est finitaire et donc $c[x]_{\equiv} = x$ pour un nombre cofini de variables x .

On montre que par induction bien fondée sur t, t' pour l'ordre produit induit par $\sqsubset \circledast \equiv$ que $t \equiv t'$ implique $t\sigma_{c,\equiv} = t'\sigma_{c,\equiv}$.

Cas $c([t]_{\equiv}), c([t']_{\equiv}) \notin X$: alors $c([t]_{\equiv}) = f(t_1, \dots, t_m)$ et $c([t']_{\equiv}) = g(t_1, \dots, t_n)$.

Si $t \equiv t'$ alors $c([t]_{\equiv}) \equiv c([t']_{\equiv})$, et comme \equiv est homogène, $f = g$, $m = n$, et $t_i \equiv t'_i$ pour tout $1 \leq i \leq m$. Par hypothèse d'induction, applicable car $t \equiv c([t]_{\equiv}) \sqsubset t_i$ et $t' \equiv c([t']_{\equiv}) \sqsubset t'_i$ pour tout $1 \leq i \leq m$, $t_i\sigma_{c,\equiv} = t'_i\sigma_{c,\equiv}$ pour tout $1 \leq i \leq m$ et donc $t\sigma_{c,\equiv} = t'\sigma_{c,\equiv}$.

Cas $c([t]_{\equiv}) \in X$ et $c([t']_{\equiv}) \notin X$: alors $t \not\equiv t'$ par définition de c .

Cas $c([t]_{\equiv}) \notin X$ et $c([t']_{\equiv}) \in X$: symétrique du cas précédent.

Cas $c([t]_{\equiv}), c([t']_{\equiv}) \in X$: alors $c([t]_{\equiv}) = x = c([x]_{\equiv})$ et $c([t']_{\equiv}) = x' = c([x']_{\equiv})$.

Alors $t \equiv t'$ implique $[t]_{\equiv} = [t']_{\equiv}$ et donc $t\sigma_{c,\equiv} = x = x' = t'\sigma_{c,\equiv}$. \square

Propriété 3.19. Si $\equiv \subseteq \cong$ sont valides et c est un choix de représentants canoniques pour \equiv , alors il existe c' un choix pour \cong tel que $\sigma_{c,\equiv} \lesssim \sigma_{c',\cong}$.

Démonstration. Pour tout $x \in X$ tel que $c([x]_{\equiv}) \in X$ et $[x]_{\cong} \setminus X \neq \emptyset$, on choisit $c'([x]_{\cong}) \in [x]_{\cong} \setminus X$. Dans tous les autres cas on pose $c'([t]_{\cong}) \stackrel{\text{def}}{=} c([t]_{\equiv})$.

On définit ensuite pour tout $x \in X$

$$\tau(x) \stackrel{\text{def}}{=} \begin{cases} c'([x]_{\cong}) & \text{si } c([x]_{\equiv}) = x \neq c'([x]_{\cong}) \\ x & \text{sinon.} \end{cases}$$

C'est bien une substitution puisque \cong est valide. Il reste à vérifier que $\sigma_{c',\cong} = \sigma_{c,\equiv}\tau$. Il suffit de vérifier que $\sigma_{c',\cong}(x) = \tau(\sigma_{c,\equiv}(x))$ pour tout $x \in X$ tel que $c([x]_{\equiv}) \in X$ et $[x]_{\cong} \setminus X \neq \emptyset$. Mais alors $c([x]_{\equiv}) = y = c([y]_{\equiv})$ et $\tau(y) = c'([y]_{\cong})$ par définition, et $x \equiv y$ implique $x \cong y$ et donc $c'([y]_{\cong}) = c'([x]_{\cong})$ comme désiré. \square

Lemme de HUET. Les propriétés précédentes permettent de déduire le résultat suivant.

Lemme 3.20 (HUET). Deux termes t et t' de $T(\mathcal{F}, X)$ sont unifiables si et seulement s'il existe une relation d'équivalence valide telle que $t \equiv t'$, auquel cas il en existe une minimale.

Démonstration. La première partie de l'énoncé découle des propriétés 3.16 et 3.18. La seconde découle du fait que les relations d'équivalence valides forment un treillis complet par la propriété 3.15. \square

De plus, par les propriétés 3.17 et 3.19, le $\text{mgu}(t \stackrel{?}{=} t')$ n'est autre que la substitution associée à cette équivalence valide minimale, pour un certain choix de représentants canoniques – mais les mgu ne sont uniques qu'à renommage par \sim près.

Remarque 3.21 (congruence). Soient les deux termes $t_1 \stackrel{\text{def}}{=} f(f(x_1, x_2), f(x_3, x_4))$ et $t_2 \stackrel{\text{def}}{=} f(f(x_2, x_3), f(x_4, x_1))$ et l'instance du problème d'unification $t_1 \stackrel{?}{=} t_2$. Un mgu est $\sigma = [x_1/x_2, x_1/x_3, x_1/x_4]$ qui rend $f(x_1, x_2)$, $f(x_2, x_3)$, $f(x_3, x_4)$ et $f(x_4, x_1)$ tous équivalents par \equiv_{σ} . Cependant, la plus petite relation d'équivalence valide \equiv telle que $t_1 \equiv t_2$ a bien $f(x_1, x_2) \equiv f(x_2, x_3)$ et $f(x_3, x_4) \equiv f(x_4, x_1)$ par homogénéité, mais $f(x_1, x_2) \not\equiv f(x_3, x_4)$.

Plus généralement, pour toute substitution σ , \equiv_σ est une *congruence* : $t_i \equiv_\sigma t'_i$ pour $1 \leq i \leq m$ implique $f(t_1, \dots, t_m) \equiv_\sigma f(t'_1, \dots, t'_m)$ pour tout $f \in \mathcal{F}_m$. On a alors une réciproque de la propriété 3.18 : si \equiv est une congruence valide, alors $t\sigma_{c,\equiv} = t'\sigma_{c,\equiv}$ implique $t \equiv t'$.

Enfin, si \equiv est une relation d'équivalence valide et \cong la plus petite congruence telle que $\equiv \subseteq \cong$, alors \cong est valide et de plus on peut utiliser le même choix c de représentants canoniques pour les deux relations et $\sigma_{c,\equiv} = \sigma_{c,\cong}$.

■ (BAADER et NIPKOW, 1998, sec. 4.8) pour ce premier algorithme.

3.4.2. *Un premier algorithme d'unification efficace.* Le principe de ce premier algorithme d'unification de t et t' est

- (1) de mettre t et t' sous la forme d'un unique graphe dirigé acyclique où chaque variable n'apparaît qu'une seule fois ;
- (2) de construire, s'il en existe une, la relation d'équivalence finitaire homogène minimale \equiv telle que $t \equiv t'$;
- (3) de tester si \equiv est acyclique : si elle ne l'est pas, alors aucune relation plus grande ne peut l'être non plus donc il n'existe pas de relation valide pour laquelle t et t' sont équivalents.

On représente les classes d'équivalence avec les structures d'`union-find`. En particulier, le choix d'un représentant canonique pour chaque classe d'équivalence découle naturellement des structures de données mises en place par `union-find`. L'algorithme fonctionne en temps déterministe $O(n \lg n)$ où $n \stackrel{\text{def}}{=} |t| + |t'|$ est la taille de la représentation des termes d'entrée, qui peuvent être fournis sous forme arborescente ou de graphes dirigés acycliques.

Étape (1). Cette étape est typiquement implémentée à l'aide d'une table de hachage. Si l'on fait l'hypothèse d'un hachage parfait, cette étape est en $O(n)$.

Par exemple, le graphe de gauche de la figure 3.2 montre le résultat de l'étape (1) pour l'entrée $f(x, y) \stackrel{?}{=} f(g(y), g(z))$, si l'on ignore les arcs en violet.

Étape (2). Cette étape commence par initialiser son entrée en ajoutant un pointeur « parent » de chaque nœud du graphe vers lui-même : ces pointeurs sont indiqués en violet dans la figure 3.2 et dénotés par $x.p$ dans le pseudo-code. Le représentant canonique d'une classe d'équivalence est obtenu en suivant ces pointeurs jusqu'à trouver un élément qui pointe sur lui-même.

Cette étape fait appel aux procédures `union` et `find` définies ci-après : il s'agit ici d'une union naïve de classes d'équivalence et d'une recherche avec compression de chemins du représentant canonique d'une classe d'équivalence.

Procédure `naive-union`(x, y)

1 $y.p := x$

Procédure `find-and-compress`(x)

1 **si** $x.p \neq x$ **alors**

2 $\lfloor x.p := \text{find-and-compress}(x.p)$

3 **retourner** $x.p$

La complexité de cette implémentation d'`union-find` est connue.

■ (BEAUQUIER, BERSTEL et PHILIPPE, 1992, sec. 3.5), (CORMEN et al., 2009, ch. 21). Les procédures d'`union-find` en tant que telles sont au programme MPI, section 3.3.

Théorème 3.22 (TARJAN et LEEUWEN, 1984, thm. 4). *L’algorithme union-find avec compression de chemin et union naïve sur n éléments effectue m appels à find en temps $\Theta(n + m \log_{2+m/n} n)$.*

L’étape (2) calcule ensuite récursivement la plus petite relation homogène qui rend t et t' équivalents. Par le théorème 3.22, ce calcul termine en $O(n \lg n)$.

Procédure `homogenise(t, t)`

```

1 t := find(t)
2 t' := find(t')
3 si t = f(t1, ..., tm), t' = g(t'1, ..., t'n) et f ≠ g alors          (non homogène!)
4 | échec : t et t' non unifiables
5 sinon si t = f(t1, ..., tm) et t' = f(t'1, ..., t'm) alors
6 | union(t, t')
7 | pour i = 1 à m faire homogenise(ti, t'i)
8 sinon si t ∈ X alors
9 | union(t', t)
10 sinon                                                                    (nécessairement t' ∈ X)
11 | union(t, t')
```

▲ À la ligne 8, il est nécessaire de rendre t' représentant canonique pour la classe contenant $t \in X$ si $t' \notin X$; de même à la ligne 10 avec les rôles de t et t' échangés. Cela empêche d'utiliser les techniques d'union par rang ou par taille; voir la section 3.4.3 pour une solution.

Exemple 3.23. La figure 3.2 présente la seconde étape de l’algorithme d’unification sur l’entrée $f(x, y) \stackrel{?}{=} f(g(y), g(z))$ déjà rencontrée dans l’exemple 3.12. Le mgu est $[g(g(z))/x, g(z)/y, z/z]$, lu en suivant les représentants canoniques de chaque variable.



FIGURE 3.2. Étape (2) de l’algorithme d’unification sur les termes d’entrée $f(x, y)$ et $f(g(y), g(z))$.

Exemple 3.24. La figure 3.3 présente la seconde étape de l’algorithme d’unification sur l’entrée $\bullet(x_1, x_2, x_3) \stackrel{?}{=} \bullet(f(x_2, x_2), f(x_3, x_3), f(x_4, x_4))$, qui encode l’exemple 3.14. On retrouve bien le même résultat, mais la représentation du mgu est linéaire en la taille de l’entrée.

Exemple 3.25. La figure 3.4 présente la seconde étape de l’algorithme d’unification sur l’entrée $f(x, y) \stackrel{?}{=} f(g(y), g(z))$. Le résultat est la substitution $[g(z)/x, g(z)/y, z/z]$, lue en suivant les représentants canoniques de chaque variable.

On peut remarquer sur cette exemple que, lors de l’appel `union(g, y)` par `homogenise(y, g)`, la classe d’équivalence de y était $\{x, y\}$, plus grande pour les

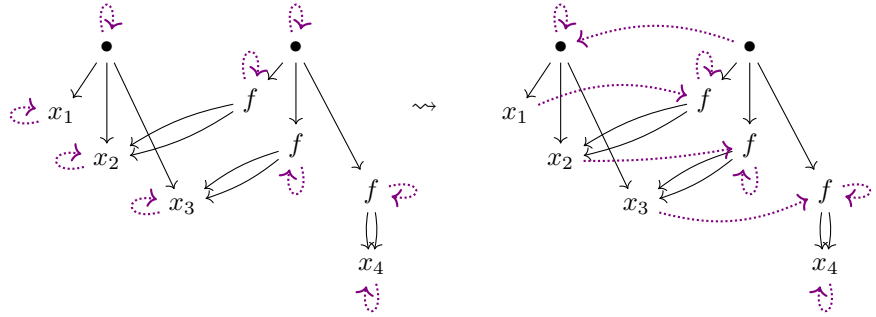


FIGURE 3.3. Étape (2) de l'algorithme d'unification sur les termes d'entrée $\bullet(x_1, x_2, x_3)$ et $\bullet(f(x_2, x_2), f(x_3, x_3), f(x_4, x_4))$.

algorithmes d'union par rang ou par taille que la classe de g qui était $\{g\}$. Nous avons cependant été contraints de choisir g comme représentant canonique.



FIGURE 3.4. Étape (2) de l'algorithme d'unification sur les termes d'entrée $f(x, y)$ et $f(y, g(z))$.

Étape (3). La dernière étape est de vérifier si la relation d'équivalence obtenue est bien acyclique. Il s'agit d'un simple test d'acyclicité dans le graphe obtenu, où l'on cherche un cycle non trivial utilisant les pointeurs « parents » et la relation de sous-terme. Ce test s'implémente en $O(n)$ par un parcours de graphe.

Exemple 3.26. La figure 3.5 présente la seconde étape de l'algorithme d'unification sur l'entrée $f(x, y) \stackrel{?}{=} f(g(y), g(x))$ déjà rencontrée dans l'exemple 3.13. La troisième étape échoue car il y a un cycle de x à lui-même utilisant au moins un arc de relation sous-terme; les termes n'étaient en effet pas unifiables.

■ (HUET, 1976, sec. 5.7) pour cet algorithme d'unification rapide.

3.4.3. Algorithme d'unification de HUET. Des améliorations de ce premier algorithme permettant une union par rang ou par taille sont possibles, avec une complexité finale en $O(n \cdot \alpha(n, n))$ où α est la fonction d'ACKERMANN inverse. Nous allons voir la version donnée par HUET. Les trois étapes de l'algorithme restent les mêmes, mais la structure sous-jacente pour union-find à l'étape (2) est différente, et les procédures union et find doivent être mises à jour.



FIGURE 3.5. Étape (2) de l'algorithme d'unification sur les termes d'entrée $f(x, y)$ et $f(g(y), g(x))$.

Structure de données. En plus du pointeur « parent » $x.p$, chaque nœud a aussi un booléen $x.r$ indiquant s'il est à la racine de la structure d'union-find; au début de l'étape (2), ce booléen est initialement 1. Le représentant canonique d'une classe est maintenant le nœud pointé par $x.p$ tel que $x.r = 1$. Chaque nœud possède également un attribut de « taille » $x.t$, initialement 1.

Procédure find. La procédure find-and-compress fait appel à une procédure auxiliaire root pour retourner le nœud racine; find-and-compress retourne alors le parent du nœud racine, qui est le représentant canonique de la classe.

Procédure root(x)

- 1 **si** $\neg x.r$ **alors**
- 2 $x.p := \text{root}(x.p)$
- 3 **retourner** $x.p$

Procédure find-and-compress(x)

- 1 **retourner** root(x). p

Procédure union. La procédure union-by-size reçoit en argument des nœuds x et y qui sont les *représentants canoniques* de leurs classes respectives, mais pas nécessairement les racines de leurs classes. On commence donc par chercher ces racines aux lignes 1 et 2. La nouvelle racine sera la racine de la classe de plus grande taille (voir les lignes 4 et 8). Enfin, le représentant canonique de la nouvelle classe sera pointée par cette racine de l'union (rien à faire aux lignes 4–6, mais voir la ligne 9).

Procédure union-by-size(x, y)

- 1 $x := \text{root}(x)$
- 2 $y := \text{root}(y)$
- 3 **si** $x.t \geq y.t$ **alors**
- 4 $y.r := 0$
- 5 $y.p := x$
- 6 $x.t := x.t + y.t$
- 7 **sinon**
- 8 $x.r := 0$
- 9 $x.p := y$
- 10 $y.p := x$
- 11 $y.t := x.t + y.t$

Exemple 3.27. Reprenons l'exemple 3.25. La figure 3.6 présente la seconde étape de l'algorithme d'unification de HUET sur l'entrée $f(x, y) \stackrel{?}{=} f(g(y), g(z))$.

Initialement, tous les nœuds ont $x.r = 1$, et sont indiqués en orange. Les tailles $x.t$ sont indiquées en vert à côté de chaque nœud.

Le résultat est la substitution $[g(z)/x, g(z)/y, z/z]$, lue en suivant dans la figure de droite les représentants canoniques de chaque variable : on remonte jusqu'à la racine de la classe indiquée en orange, et on suit son lien parent en violet pour trouver le représentant canonique.



FIGURE 3.6. Étape (2) de l'algorithme d'unification de HUET sur les termes d'entrée $f(x, y)$ et $f(y, g(z))$.

Exemple 3.28. Reprenons l'exemple 3.26. La figure 3.7 présente la seconde étape de l'algorithme d'unification de HUET sur l'entrée $f(x, y) \stackrel{?}{=} f(g(y), g(x))$. La troisième étape échoue comme il se doit, car il y a un cycle de x à lui-même utilisant au moins un arc de relation sous-terme ; les termes n'étaient en effet pas unifiables.



FIGURE 3.7. Étape (2) de l'algorithme d'unification de HUET sur les termes d'entrée $f(x, y)$ et $f(g(y), g(x))$.

La complexité de cette implémentation d'`union-find` est elle aussi connue, et donne au final une complexité en $O(n \cdot \alpha(n, n))$ pour l'algorithme d'unification de HUET.

■ Voir (BEAUQUIER, BERSTEL et PHILIPPE, 1992, thm. 5.3), ou (CORMEN et al., 2009, thm. 21.14) pour une version avec union par rang.

Théorème 3.29. L'algorithme `union-find` avec compression de chemin et union par taille sur n éléments effectue m appels à `find` en temps $\Theta(m \cdot \alpha(m, n))$.

Partie 2

Sémantique de vérité du calcul propositionnel

Programme officiel. Par souci d'éviter trop de technicité, on ne présente la notion de valeur de vérité que pour des formules sans quantificateurs.

<i>Notions</i>	<i>Commentaires</i>
Valuations, valeurs de vérité d'une formule propositionnelle.	Notations V pour la valeur vraie, F pour la valeur fausse.
Satisfiabilité, modèle, ensemble de modèles, tautologie, antilogie.	Une formule est satisfiable si elle admet un modèle, tautologique si toute valuation en est un modèle. On peut être amené à ajouter à la syntaxe une formule tautologique et une formule antilogique; elles sont en ce cas notées \top et \perp .
Équivalence sur les formules.	On présente les lois de DE MORGAN, le tiers exclu et la décomposition de l'implication.
Conséquence logique entre deux formules.	On étend la notion à celle de conséquence φ d'un ensemble de formules Γ : on note $\Gamma \models \varphi$. La compacité est hors programme.
Forme normale conjonctive, forme normale disjonctive.	Lien entre forme normale disjonctive complète et table de vérité.
Mise sous forme normale.	On peut représenter les formes normales comme des listes de listes de littéraux. Exemple de formule dont la taille des formes normales est exponentiellement plus grande.
Problème SAT, k SAT, algorithme de QUINE.	On incarne SAT par la modélisation d'un problème (par exemple la coloration des sommets d'un graphe).

Leçon. Cette partie est aussi au programme de l'épreuve d'oral de leçon de l'agrégation.

Leçon 28. Formules du calcul propositionnel : représentation, formes normales, satisfiabilité. Applications.

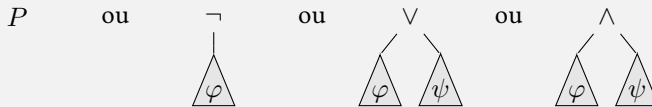
Nous faisons un rapide rappel de la syntaxe et de la sémantique de la logique propositionnelle dans les sections 4 et 5; il s'agit d'une reformulation de la partie 1 pour le cas propositionnel. Les équivalences et conséquences logiques sont vues dans la section 6 et le théorème de compacité en section 6.5.

Les formes normales sont présentées dans la section 7; en particulier le traitement du coût exponentiel de la mise sous forme normale se trouve en section 7.3.2. En guise de supplément pour la leçon 28, on présente aussi rapidement les formes disjonctives premières et minimales en sections 7.3.2 à 7.3.3 et les diagrammes binaires de décision en section 7.4.

Le problème de satisfiabilité et des exemples de modélisation sont donnés en section 8. L'algorithme de QUINE de recherche de modèle par simplification est donné en section 8.3.2. En guise de supplément pour la leçon **28**, on présente aussi l'algorithme de recherche de modèle dû à DAVIS, PUTNAM, LOGEMANN et LOVELAND en section 8.3.3.

4. SYNTAXE

Résumé. Étant donné un ensemble dénombrable \mathcal{P}_0 de *propositions*, les formules propositionnelles sont des arbres dont les feuilles sont des propositions et les nœuds internes des connecteurs logiques. Ainsi, une *formule propositionnelle* est un arbre de la forme



où $P \in \mathcal{P}_0$ et φ et ψ sont des formules propositionnelles.

4.1. Arbres de syntaxe abstraite. Soit \mathcal{P}_0 un ensemble infini dénombrable de symboles de propositions (aussi appelés « variables propositionnelles »). La syntaxe de la logique propositionnelle est définie par la syntaxe abstraite

$$\varphi ::= P \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \quad (\text{formules propositionnelles})$$

où $P \in \mathcal{P}_0$.

Concrètement, cela signifie qu'une formule propositionnelle est un *arbre fini*, dont les feuilles sont étiquetées par des propositions tirées de \mathcal{P}_0 , et dont les nœuds internes sont étiquetés soit par \neg (pour « non ») et ont exactement un nœud enfant, soit par \vee (pour « ou ») ou \wedge (pour « et ») et ont exactement deux nœuds enfants. Par exemple, la figure 4.1 décrit une formule propositionnelle φ_{ex} où P et Q sont des propositions de \mathcal{P}_0 .

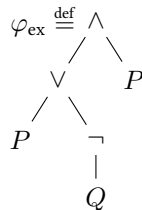


FIGURE 4.1. Une formule propositionnelle.

■ (DUPARC, 2015, sec. I.1.2),
(DAVID, NOUR et RAFFALLI, 2003,
sec. 1.2.6), (GOUBAULT-LARRECQ
et MACKIE, 1997, sec. 2.1),
(HARRISSON, 2009, sec. 2.1)

L'intérêt de travailler avec des arbres de syntaxe abstraite est que ceux-ci se prêtent très bien aux définitions et aux algorithmes par récurrence. Par exemple, on peut définir l'ensemble $\text{Props}(\varphi) \subseteq \mathcal{P}_0$ des propositions qui apparaissent au moins une fois dans une formule propositionnelle φ :

$$\begin{aligned} \text{Props}(P) &\stackrel{\text{def}}{=} \{P\} , & \text{Props}(\neg\varphi) &\stackrel{\text{def}}{=} \text{Props}(\varphi) , \\ \text{Props}(\varphi \vee \psi) &\stackrel{\text{def}}{=} \text{Props}(\varphi) \cup \text{Props}(\psi) & \text{Props}(\varphi \wedge \psi) &\stackrel{\text{def}}{=} \text{Props}(\varphi) \cup \text{Props}(\psi). \end{aligned}$$

Dans l'exemple de la figure 4.1, $\text{Props}(\varphi_{\text{ex}}) = \{P, Q\}$.

■ (DUPARC, 2015, sec. I.1.4)

4.2. Syntaxe concrète. Cependant, dessiner des arbres à chaque fois que l'on veut écrire une formule propositionnelle est plutôt laborieux. On utilise plutôt une écriture « linéaire » en introduisant des parenthèses de manière judicieuse. Par exemple, la formule propositionnelle de la figure 4.1 s'écrit $\varphi_{\text{ex}} \stackrel{\text{def}}{=} ((P \vee \neg Q) \wedge P)$. On se permet généralement des facilités d'écritures, comme $(P \vee \neg Q) \wedge P$ pour φ_{ex} – où l'on a enlevé les parenthèses extérieures –, ou $P \wedge Q \wedge R$ pour $((P \wedge Q) \wedge R)$ ou $(P \wedge (Q \wedge R))$ – car les opérateurs \wedge et \vee sont associatifs (voir section 6).

5. SÉMANTIQUE

Résumé. Soit $\mathbb{B} \stackrel{\text{def}}{=} \{\mathbf{F}, \mathbf{V}\}$ l'ensemble des *valeurs de vérité*, où \mathbf{F} désigne « faux » et \mathbf{V} désigne « vrai ». Étant donnée une *interprétation* $I: \mathcal{P}_0 \rightarrow \mathbb{B}$, la *sémantique* $\llbracket \varphi \rrbracket^I$ d'une formule propositionnelle est une valeur de vérité, qui ne dépend en réalité que des propositions qui apparaissent dans φ (propriété 5.3). On note « $I \models \varphi$ » si $\llbracket \varphi \rrbracket^I = \mathbf{V}$.

Ainsi, on peut aussi voir $\llbracket \varphi \rrbracket$ comme une *fonction booléenne* qui prend en argument les valeurs de vérité de ses propositions et retourne une valeur de vérité, et pour laquelle on peut écrire une *table de vérité*. Inversement, pour toute fonction booléenne \mathbf{f} , il existe une formule propositionnelle φ telle que $\mathbf{f} = \llbracket \varphi \rrbracket$ (théorème 5.8 de complétude fonctionnelle).

Une formule propositionnelle φ est *satisfiable* s'il existe une interprétation I telle que $\llbracket \varphi \rrbracket^I = \mathbf{V}$. Elle est *valide* (noté « $\models \varphi$ ») si pour toute interprétation I , on a $\llbracket \varphi \rrbracket^I = \mathbf{V}$.

5.1. Valeurs de vérité. On note $\mathbb{B} \stackrel{\text{def}}{=} \{\mathbf{F}, \mathbf{V}\}$ pour l'ensemble des *valeurs de vérité* : l'intuition est que \mathbf{V} dénote la valeur « vrai » et \mathbf{F} la valeur « faux ». On définit aussi trois opérations $\text{non} : \mathbb{B} \rightarrow \mathbb{B}$ et $\text{et}, \text{ou} : \mathbb{B}^2 \rightarrow \mathbb{B}$ définies par $\text{non } \mathbf{V} = \mathbf{F}$ ou $\mathbf{F} = \mathbf{F}$ et $\mathbf{F} = \mathbf{V}$ et $\mathbf{F} = \mathbf{F}$ et $\mathbf{V} = \mathbf{F}$ et $\text{non } \mathbf{F} = \mathbf{V}$ ou $\mathbf{V} = \mathbf{V}$ ou $\mathbf{F} = \mathbf{F}$ ou $\mathbf{V} = \mathbf{V}$ et $\mathbf{V} = \mathbf{V}$ (voir la table 5.1). On appelle aussi \mathbb{B} muni des trois opérations non , ou et et « l'algèbre de BOOLE ».

Remarque 5.1. En français, le mot « ou » est ambigu, dans la mesure où il peut être compris de manière *inclusive* (A ou B ou les deux) ou *exclusive* (A ou B mais pas les deux). Dans l'usage courant, les deux cas A et B sont souvent implicitement exclusifs (« Je me lève à sept ou huit heures selon les jours »). Le « **ou** » logique est en revanche inclusif.

■ (DUPARC, 2015, sec. I.2.3)

5.1.1. Interprétations. Une *interprétation* est une fonction (aussi appelée une « valuation propositionnelle ») $I: \mathcal{P}_0 \rightarrow \mathbb{B}$ qui associe une valeur de vérité $P^I \in \mathbb{B}$ pour chaque proposition $P \in \mathcal{P}_0$. Si I est une interprétation et P est une proposition, on écrit $I[\mathbf{V}/P]$ (resp. $I[\mathbf{F}/P]$) pour l'interprétation qui associe \mathbf{V} à P (resp. \mathbf{F}) et Q^I à Q pour tout $Q \neq P$.

■ (DUPARC, 2015, sec. I.2.1)

☞ On peut aussi voir une interprétation comme un sous-ensemble $I \stackrel{\text{def}}{=} \{P \in \mathcal{P}_0 \mid P^I = \mathbf{V}\}$ de propositions dans $2^{\mathcal{P}_0}$; les deux points de vue sont bien sûr équivalents.

■ (DUPARC, 2015, sec. I.2.2),

(GOUBAULT-LARRECQ et

MACKIE, 1997, def. 2.8),

(HARRISSON, 2009, sec. 2.2)

5.1.2. Sémantique. La sémantique $\llbracket \varphi \rrbracket^I \in \mathbb{B}$ d'une formule propositionnelle φ dans une interprétation I est définie inductivement par

$$\llbracket P \rrbracket^I \stackrel{\text{def}}{=} P^I, \quad \llbracket \neg \varphi \rrbracket^I \stackrel{\text{def}}{=} \text{non } \llbracket \varphi \rrbracket^I, \quad \llbracket \varphi \vee \psi \rrbracket^I \stackrel{\text{def}}{=} \llbracket \varphi \rrbracket^I \text{ ou } \llbracket \psi \rrbracket^I, \quad \llbracket \varphi \wedge \psi \rrbracket^I \stackrel{\text{def}}{=} \llbracket \varphi \rrbracket^I \text{ et } \llbracket \psi \rrbracket^I$$

On dit que I *satisfait* φ (ou que I est un « modèle » de φ), noté $I \models \varphi$, si $\llbracket \varphi \rrbracket^I = \mathbf{V}$; cette écriture peut être définie de manière équivalente par

$$\begin{aligned} I \models P & & \text{si } P \in I, \\ I \models \neg\varphi & & \text{si } I \not\models \varphi, \\ I \models \varphi \vee \psi & & \text{si } I \models \varphi \text{ ou } I \models \psi, \\ I \models \varphi \wedge \psi & & \text{si } I \models \varphi \text{ et } I \models \psi. \end{aligned}$$

On définit aussi $\text{Sat}(\varphi) \stackrel{\text{def}}{=} \{I \in \mathbb{B}^{\mathcal{P}_0} \mid I \models \varphi\}$ l'ensemble des interprétations qui satisfont φ .

Exemple 5.2. Considérons à nouveau la formule propositionnelle $\varphi_{\text{ex}} = (P \vee \neg Q) \wedge P$ de la figure 4.1, et l'interprétation I qui associe $P^I = \mathbf{F}$ à P , $Q^I = \mathbf{F}$ à Q , et $R^I = \mathbf{V}$ à toutes les propositions $R \in \mathcal{P}_0$ différentes de P et Q . La sémantique $\llbracket \varphi_{\text{ex}} \rrbracket^I$ se calcule comme suit :

$$\begin{aligned} \llbracket (P \vee \neg Q) \wedge P \rrbracket^I &= \llbracket P \vee \neg Q \rrbracket^I \text{ et } \llbracket P \rrbracket^I \\ &= (\llbracket P \rrbracket^I \text{ ou } \llbracket \neg Q \rrbracket^I) \text{ et } \llbracket P \rrbracket^I \\ &= (P^I \text{ ou } \llbracket \neg Q \rrbracket^I) \text{ et } \llbracket P \rrbracket^I \\ &= (\mathbf{F} \text{ ou } \llbracket \neg Q \rrbracket^I) \text{ et } \llbracket P \rrbracket^I \\ &= (\mathbf{F} \text{ ou non } \llbracket Q \rrbracket^I) \text{ et } \llbracket P \rrbracket^I \\ &= (\mathbf{F} \text{ ou non } Q^I) \text{ et } \llbracket P \rrbracket^I \\ &= (\mathbf{F} \text{ ou non } \mathbf{F}) \text{ et } \llbracket P \rrbracket^I \\ &= (\mathbf{F} \text{ ou } \mathbf{V}) \text{ et } \llbracket P \rrbracket^I \\ &= \mathbf{V} \text{ et } \llbracket P \rrbracket^I \\ &= \mathbf{V} \text{ et } P^I \\ &= \mathbf{V} \text{ et } \mathbf{F} \\ &= \mathbf{F}. \end{aligned}$$

À noter que l'on aurait pu atteindre ce résultat bien plus vite en écrivant

$$\begin{aligned} \llbracket (P \vee \neg Q) \wedge P \rrbracket^I &= \llbracket P \vee \neg Q \rrbracket^I \text{ et } \llbracket P \rrbracket^I \\ &= \llbracket P \vee \neg Q \rrbracket^I \text{ et } P^I \\ &= \llbracket P \vee \neg Q \rrbracket^I \text{ et } \mathbf{F} \\ &= \mathbf{F}, \end{aligned}$$

sans se préoccuper d'évaluer $\llbracket P \vee \neg Q \rrbracket^I$.

On peut observer que la valeur de vérité de φ ne dépend que de l'interprétation des propositions de $\text{Props}(\varphi)$: si $P \notin \text{Props}(\varphi)$, alors pour toute interprétation I , $\llbracket \varphi \rrbracket^{I[V/P]} = \llbracket \varphi \rrbracket^{I[F/P]}$. Cela se démontre par induction structurelle sur φ comme suit.

Propriété 5.3. Pour toute formule propositionnelle φ et interprétations I et I' , si $P^I = P^{I'}$ pour toute proposition $P \in \text{Props}(\varphi)$, alors $\llbracket \varphi \rrbracket^I = \llbracket \varphi \rrbracket^{I'}$.

Démonstration. Par induction structurelle sur φ .

☞ On peut aussi définir la sémantique d'une formule propositionnelle via un jeu d'évaluation; voir (DUPARC, 2015, sec. I.2.7).

■ (HARRISSON, 2009, thm. 2.2)

Pour le cas de base $\varphi = P$, on a $P \in \text{Props}(\varphi)$ donc $P^I = P^{I'}$ par hypothèse sur I et I' , et on a bien

$$\llbracket P \rrbracket^I = P^I = P^{I'} = \llbracket P \rrbracket^{I'} .$$

Pour l'étape d'induction, si $\varphi = \neg\psi$, on a $\text{Props}(\varphi) = \text{Props}(\psi)$, donc $P^I = P^{I'}$ pour toute proposition $P \in \text{Props}(\psi)$ et on peut appliquer l'hypothèse d'induction à la sous-formule ψ : on a bien

$$\llbracket \neg\psi \rrbracket^I = \text{non } \llbracket \psi \rrbracket^I \stackrel{\text{ih}}{=} \text{non } \llbracket \psi \rrbracket^{I'} = \llbracket \neg\psi \rrbracket^{I'} .$$

Si $\varphi = \psi \vee \psi'$, on a $\text{Props}(\varphi) = \text{Props}(\psi) \cup \text{Props}(\psi')$, donc $P^I = P^{I'}$ pour toute proposition $P \in \text{Props}(\psi)$ ou $P \in \text{Props}(\psi')$ et on peut appliquer l'hypothèse d'induction aux deux sous-formules ψ et ψ' : on a bien

$$\llbracket \psi \vee \psi' \rrbracket^I = \llbracket \psi \rrbracket^I \text{ ou } \llbracket \psi' \rrbracket^I \stackrel{\text{ih}}{=} \llbracket \psi \rrbracket^{I'} \text{ ou } \llbracket \psi' \rrbracket^{I'} = \llbracket \psi \vee \psi' \rrbracket^{I'} .$$

Enfin, le cas où $\varphi = \psi \wedge \psi'$ est similaire au précédent. \square

La propriété 5.3 signifie que, pour une interprétation I et une formule propositionnelle φ , seules les valeurs de vérité P^I pour $P \in \text{Props}(\varphi)$ influencent la sémantique $\llbracket \varphi \rrbracket^I$. On pourrait donc aussi employer des interprétations *partielles* $I : \mathcal{P}_0 \rightarrow \mathbb{B}$ pour peu que leur domaine $\text{dom}(I)$ contienne les propositions de $\text{Props}(\varphi)$.

Introduisons quelques notations supplémentaires pour ces interprétations partielles. Pour des propositions distinctes P_1, \dots, P_n et des valeurs de vérité b_1, \dots, b_n dans \mathbb{B} , on note $[b_1/P_1, \dots, b_n/P_n]$ pour l'interprétation partielle de domaine fini $\{P_1, \dots, P_n\}$ telle que $P_j^{[b_1/P_1, \dots, b_n/P_n]} \stackrel{\text{def}}{=} b_j$ pour tout $1 \leq j \leq n$. Pour deux interprétations partielles I et I' , on dit que I' *étend* I ou que I est la *restriction* de I' à $\text{dom}(I)$, et on écrit $I \sqsubseteq I'$, si $\text{dom}(I) \subseteq \text{dom}(I')$ et pour toute proposition $P \in \text{dom}(I)$, $P^I = P^{I'}$.

Exemple 5.4. Comme vu dans l'exemple 5.2, on a $\llbracket \varphi_{\text{ex}} \rrbracket^I = \mathbf{F}$ pour toute interprétation I qui étend l'interprétation partielle $[F/P, F/Q]$.

5.1.3. *Tables de vérité.* Les opérateurs **non**, **ou** et **et** sont des *fonctions booléennes*, c'est-à-dire des fonctions $\mathbf{f} : \mathbb{B}^n \rightarrow \mathbb{B}$ pour un certain $n > 0$. Une façon de présenter de telles fonctions est sous la forme de *tables de vérité*, où chaque ligne indique les valeurs possibles des arguments x_1, \dots, x_n de la fonction ainsi que la valeur de $\mathbf{f}(x_1, \dots, x_n)$; voir la table 5.1 pour les tables de vérité de **non**, **ou** et **et**.

■ (DUPARC, 2015, sec. I.2.8), (JAUME et al., 2020, sec. 7.3.1); les fonctions booléennes apparaissent aussi dans la leçon 19 « Fonctions et circuits booléens en architecture des ordinateurs ».

TABLE 5.1. Les tables de vérité des fonctions **non**, **ou** et **et**.

x_1	$\text{non } x_1$	x_1	x_2	$x_1 \text{ ou } x_2$	x_1	x_2	$x_1 \text{ et } x_2$
V	F	V	V	V	V	V	V
V	F	V	F	V	V	F	F
F	V	F	V	V	F	V	F
F	V	F	F	F	F	F	F

Soit φ une formule propositionnelle. Par la propriété 5.3, la satisfaction de φ ne dépend que de l'interprétation des propositions de $\text{Props}(\varphi)$. On peut ainsi voir φ comme définissant une fonction des interprétations partielles $I \in \mathbb{B}^{\text{Props}(\varphi)}$ dans \mathbb{B} : $\llbracket \varphi \rrbracket(I) \stackrel{\text{def}}{=} \llbracket \varphi \rrbracket^I$. C'est donc

une fonction booléenne $\mathbb{B}^n \rightarrow \mathbb{B}$ à $n = |\text{Props}(\varphi)|$ variables, et on peut en donner la table de vérité. Cependant, plutôt que de seulement donner la table de vérité de $\llbracket \varphi \rrbracket$, il peut être pratique de donner par la même occasion la table de vérité de chacune de ses sous-formules.

TABLE 5.2. La table de vérité de la formule propositionnelle φ_{ex} de la figure 4.1.

P	Q	$\neg Q$	$P \vee \neg Q$	$(P \vee \neg Q) \wedge P$
V	V	F	V	V
V	F	V	V	V
F	V	F	F	F
F	F	V	V	F

Exemple 5.5. Pour la formule propositionnelle φ_{ex} de la figure 4.1, $\llbracket \varphi_{\text{ex}} \rrbracket$ est une fonction à deux variables qui représentent les valeurs de $\llbracket P \rrbracket^I$ et $\llbracket Q \rrbracket^I$. La formule propositionnelle φ_{ex} a pour sous-formules P , Q , $\neg Q$, $P \vee \neg Q$, ainsi que $(P \vee \neg Q) \wedge P$. La table de vérité correspondante est donnée dans la table 5.2.

Les deux colonnes « $\neg Q$ » et « $P \vee \neg Q$ » contiennent des calculs intermédiaires. La colonne « $\neg Q$ » est obtenue en appliquant **non** à la colonne Q ; la colonne « $P \vee \neg Q$ » l'est en appliquant **ou** aux colonnes P et $\neg Q$; enfin, la colonne « $(P \vee \neg Q) \wedge P$ » est obtenue en appliquant **et** aux colonnes $P \vee \neg Q$ et P . À noter que la sémantique $\llbracket \varphi_{\text{ex}} \rrbracket^I = \mathbf{F}$ calculée dans l'exemple 5.2 pour I étendant $[F/P, F/Q]$ correspond à la dernière ligne de la table. On peut remarquer dans cette table que les colonnes « P » et « $(P \vee \neg Q) \wedge P$ » contiennent les mêmes valeurs de vérité : ces deux formules propositionnelles sont dites *fonctionnellement équivalentes*, car elles définissent les mêmes fonctions $\llbracket P \rrbracket = \llbracket (P \vee \neg Q) \wedge P \rrbracket$.

■ (DUPARC, 2015, sec. I.2.11)

5.1.4. * *Étendre la syntaxe.* D'autres opérations booléennes que **non**, **ou** et **et** pourraient être utilisées dans la syntaxe des formules propositionnelles. Par exemple, il y a $2^{2^2} = 16$ fonctions booléennes à deux arguments, dont les fonctions indiquées dans la table 5.3.

TABLE 5.3. Les tables de vérité des fonctions booléennes **xor** (« ou exclusif »), **impl** (« implique »), **equiv** (« si et seulement si ») et **nand** (« non et »).

x_1	x_2	$x_1 \text{ xor } x_2$	$x_1 \text{ impl } x_2$	$x_1 \text{ equiv } x_2$	$x_1 \text{ nand } x_2$
V	V	F	V	V	F
V	F	V	F	F	V
F	V	V	V	F	V
F	F	F	V	V	V

Pour chacune de ces fonctions, on pourrait étendre la syntaxe abstraite des formules propositionnelles pour s'autoriser à les utiliser :

$$\varphi ::= \dots \mid \varphi \oplus \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid \varphi \uparrow \varphi,$$

en étendant de même la sémantique par

$$\begin{aligned} \llbracket \varphi \oplus \psi \rrbracket^I &\stackrel{\text{def}}{=} \llbracket \varphi \rrbracket^I \text{ xor } \llbracket \psi \rrbracket^I, & \llbracket \varphi \rightarrow \psi \rrbracket^I &\stackrel{\text{def}}{=} \llbracket \varphi \rrbracket^I \text{ impl } \llbracket \psi \rrbracket^I, \\ \llbracket \varphi \leftrightarrow \psi \rrbracket^I &\stackrel{\text{def}}{=} \llbracket \varphi \rrbracket^I \text{ equiv } \llbracket \psi \rrbracket^I, & \llbracket \varphi \uparrow \psi \rrbracket^I &\stackrel{\text{def}}{=} \llbracket \varphi \rrbracket^I \text{ nand } \llbracket \psi \rrbracket^I. \end{aligned}$$

Remarque 5.6. Comme mentionné dans la remarque 5.1, le « ou exclusif » **xor** correspond au sens souvent implicite du mot « ou » en français (A ou B mais pas les deux).

L'implication **impl** correspond approximativement à « si A alors B » ou « A implique B ». Mais en français usuel, ces locutions établissent souvent un lien de causalité implicite entre A et B, comme dans « S'il pleut, alors je prends mon parapluie. ». Cela rend aussi une phrase comme « S'il pleut et que je mets mon pull bleu, alors il pleut. » assez étrange, alors que $(P \wedge B) \rightarrow P$ est une formule propositionnelle tout à fait raisonnable. L'implication en français courant peut aussi prendre un sens exclusif, comme dans « S'il reste ici, je m'en vais. », qu'on formaliserait en $R \oplus \neg V$ où R dénote « il reste ici » et V « je m'en vais ».

■ (DUPARC, 2015, sec. I.2.3)

5.1.5. * *Complétude fonctionnelle.* Pour les applications de la logique propositionnelle, par exemple pour les expressions booléennes dans les langages de programmation ou de circuits logiques, il serait pour le moins souhaitable que toutes les fonctions booléennes soient exprimables comme la sémantique $\llbracket \varphi \rrbracket^I$ d'une formule propositionnelle φ . Une solution serait, pour chaque fonction booléenne $f: \mathbb{B}^n \rightarrow \mathbb{B}$, d'étendre la syntaxe abstraite des formules propositionnelles par

■ (DUPARC, 2015, sec. I.2.12),
(JAUME et al., 2020, sec. 7.2.1.1)

$$\varphi ::= \dots \mid f(\varphi, \dots, \varphi)$$

et leur sémantique par

$$\llbracket f(\varphi_1, \dots, \varphi_n) \rrbracket^I \stackrel{\text{def}}{=} f(\llbracket \varphi_1 \rrbracket^I, \dots, \llbracket \varphi_n \rrbracket^I).$$

Fort heureusement, il n'est pas nécessaire d'enrichir la syntaxe et la sémantique des formules propositionnelles par une infinité de cas – ce qui serait un obstacle pour leur implémentation! En effet, toutes les fonctions booléennes peuvent être exprimées à l'aide des seules **non**, **ou** et **et** (et des fonctions de projection) en les composant de manière appropriée.

Exemple 5.7. Les fonctions de la table 5.3 s'expriment comme suit :

$$\begin{aligned} x_1 \text{ xor } x_2 &= (x_1 \text{ ou } x_2) \text{ et non } (x_1 \text{ et } x_2), & x_1 \text{ impl } x_2 &= \text{non } x_1 \text{ ou } x_2, \\ x_1 \text{ equiv } x_2 &= (\text{non } x_1 \text{ ou } x_2) \text{ et } (\text{non } x_2 \text{ ou } x_1), & x_1 \text{ nand } x_2 &= \text{non } (x_1 \text{ et } x_2). \end{aligned}$$

Intuitivement, les fonctions booléennes définissables comme des sémantiques $\llbracket \varphi \rrbracket^I$ de formules propositionnelles sont justement les fonctions exprimables à l'aide des seules **non**, **ou** et **et**; par exemple, $\text{impl} = \llbracket \neg P_1 \vee P_2 \rrbracket^I$. On obtient donc le résultat suivant.

Théorème 5.8 (complétude fonctionnelle). *Pour tout $n > 0$ et toute fonction booléenne $f: \mathbb{B}^n \rightarrow \mathbb{B}$, il existe une formule propositionnelle φ sur n propositions P_1, \dots, P_n telle que $f = \llbracket \varphi \rrbracket^I$.*

Démonstration. Soit f une fonction booléenne $\mathbb{B}^n \rightarrow \mathbb{B}$ pour un certain $n > 0$. Si $n = 1$, alors il y a quatre fonctions booléennes de \mathbb{B} dans \mathbb{B} , qui sont toutes exprimables à l'aide de formules propositionnelles :

- la fonction identité est $\llbracket P_1 \rrbracket^I$,
- la fonction négation $\llbracket \neg P_1 \rrbracket^I$,
- la fonction constante \vee est $\llbracket P_1 \vee \neg P_1 \rrbracket^I$ et

– la fonction constante **F** est $\llbracket P_1 \wedge \neg P_1 \rrbracket$.

Si $n > 1$, alors

$$f = \llbracket \bigvee_{(b_1, \dots, b_n) \in \mathbb{B}^n : f(b_1, \dots, b_n) = \mathbf{V}} \varphi_{(b_1, \dots, b_n)} \rrbracket$$

pourvu que chaque formule propositionnelle $\varphi_{(b_1, \dots, b_n)}$ soit telle que $\llbracket \varphi_{(b_1, \dots, b_n)} \rrbracket^I = \mathbf{V}$ si et seulement si pour tout $1 \leq i \leq n$, $P_i^I = b_i$. De telles formules propositionnelles $\varphi_{(b_1, \dots, b_n)}$ s'écrivent comme des conjonctions

$$\varphi_{(b_1, \dots, b_n)} \stackrel{\text{def}}{=} \bigwedge_{1 \leq i \leq n} \ell_{(b_1, \dots, b_n), i}$$

où les littéraux $\ell_{(b_1, \dots, b_n), i}$ sont définis par

$$\ell_{(b_1, \dots, b_n), i} \stackrel{\text{def}}{=} \begin{cases} P_i & \text{si } b_i = \mathbf{V} \\ \neg P_i & \text{si } b_i = \mathbf{F}. \end{cases} \quad \square$$

Exemple 5.9. Appliquons la preuve du théorème 5.8 de complétude fonctionnelle à la fonction **equiv** définie dans la table 5.3. Il y a deux valuations de (x_1, x_2) telles que x_1 **equiv** $x_2 = \mathbf{V}$, à savoir (\mathbf{V}, \mathbf{V}) et (\mathbf{F}, \mathbf{F}) . Les deux formules propositionnelles associées sont $\varphi_{(\mathbf{V}, \mathbf{V})} \stackrel{\text{def}}{=} P_1 \wedge P_2$ et $\varphi_{(\mathbf{F}, \mathbf{F})} \stackrel{\text{def}}{=} \neg P_1 \wedge \neg P_2$, et on a bien **equiv** = $\llbracket (P_1 \wedge P_2) \vee (\neg P_1 \wedge \neg P_2) \rrbracket$.

■ (DUPARC, 2015, sec. I.2.9),
(HARRISON, 2009, sec. 2.3)

5.2. Satisfiabilité et validité. Une formule propositionnelle φ est *satisfiable* s'il existe un modèle de φ , c'est-à-dire s'il existe une interprétation I telle que $I \models \varphi$. Elle est *valide*, noté $\models \varphi$, si pour toute interprétation I , $I \models \varphi$. Clairement, une formule propositionnelle valide est en particulier satisfiable, mais l'inverse n'est pas toujours vrai.

En termes des tables de vérité de la section 5.1.3, une formule propositionnelle est donc satisfiable si et seulement s'il existe au moins une entrée **V** dans sa colonne, et elle est valide si et seulement si sa colonne est entièrement constituée de **V**. On voit dans la table 5.2 que la formule propositionnelle φ_{ex} de la figure 4.1 est satisfiable mais pas valide.

En termes d'ensemble de modèles, rappelons que $\text{Sat}(\varphi) \stackrel{\text{def}}{=} \{I \in \mathbb{B}^{\mathcal{P}_0} \mid I \models \varphi\}$ dénote l'ensemble des interprétations qui satisfont φ . Alors φ est satisfiable si $\text{Sat}(\varphi) \neq \emptyset$, tandis qu'elle est valide si $\text{Sat}(\varphi) = \mathbb{B}^{\mathcal{P}_0}$ est l'ensemble de toutes les interprétations possibles.

Exemple 5.10 (loi de PEIRCE). La formule propositionnelle $\varphi \stackrel{\text{def}}{=} ((P \rightarrow Q) \rightarrow P) \rightarrow P$ est une variante du tiers exclu, et est valide en logique classique propositionnelle. En effet, soit I une interprétation quelconque. Si $I \models P$, alors $I \models \varphi$. Sinon, $I \models P \rightarrow Q$ donc $I \not\models (P \rightarrow Q) \rightarrow P$ et donc $I \models \varphi$.

Propriété 5.11 (dualité entre satisfiabilité et validité). *Une formule propositionnelle φ n'est pas satisfiable si et seulement si $\neg\varphi$ est valide; elle n'est pas valide si et seulement si $\neg\varphi$ est satisfiable.*

Démonstration. Pour le premier énoncé, φ n'est pas satisfiable (aussi dite « contradictoire »)

- si et seulement si, pour toute interprétation I , $I \not\models \varphi$,
- si et seulement si, pour toute interprétation I , $I \models \neg\varphi$,
- si et seulement si, $\neg\varphi$ est valide.

Pour le second énoncé, φ n'est pas valide (aussi dite « falsifiable »)

- si et seulement si, il existe une interprétation I telle que $I \not\models \varphi$,
- si et seulement si, il existe une interprétation I telle que $I \models \neg\varphi$,
- si et seulement si, $\neg\varphi$ est satisfiable. □

Pour un ensemble de formules propositionnelles S et une interprétation I , on écrit $I \models S$ si $I \models \psi$ pour tout $\psi \in S$. Un ensemble S est *insatisfiable* s'il n'existe pas d'interprétation I telle que $I \models S$.

Exemple 5.12 (ensemble insatisfiable). Soit l'ensemble F de formules propositionnelles suivant :

$$\{P \vee Q \vee \neg R, Q \vee R, \neg P \vee \neg Q \vee R, \neg P \vee \neg R, P \vee \neg Q\}.$$

Soit I une interprétation. Supposons tout d'abord que $I \models P$. Si $I \models R$, alors $I \not\models \neg P \vee \neg R$; sinon si $I \models Q$ alors $I \not\models \neg P \vee \neg Q \vee R$ et sinon $I \not\models Q \vee R$. Supposons maintenant $I \not\models P$. Si $I \models Q$, alors $I \not\models P \vee \neg Q$; sinon si $I \models R$ alors $I \not\models P \vee Q \vee \neg R$ et sinon $I \not\models Q \vee R$.

6. CONSÉQUENCES ET ÉQUIVALENCES LOGIQUES

Résumé. Une formule propositionnelle φ est une *conséquence logique* d'une formule propositionnelle ψ (noté « $\psi \models \varphi$ ») si pour toute interprétation I , si $I \models \psi$ alors $I \models \varphi$. C'est le cas si et seulement si la formule propositionnelle $\psi \rightarrow \varphi$ est valide, si et seulement si $\llbracket \psi \rrbracket \leq \llbracket \varphi \rrbracket$ (lemmes 6.2 et 6.3). Les formules propositionnelles φ et ψ sont *logiquement équivalentes* si $\psi \models \varphi$ et $\varphi \models \psi$; c'est le cas si et seulement si $\psi \leftrightarrow \varphi$ est valide, si et seulement si $\llbracket \psi \rrbracket = \llbracket \varphi \rrbracket$.

Une *substitution propositionnelle* est une fonction τ de domaine fini qui associe à toute proposition $P \in \mathcal{P}_0$ une formule propositionnelle $\tau(P)$; par extension, $\varphi\tau$ est la formule propositionnelle dans laquelle toutes les occurrences de chaque proposition P ont été remplacées par $\tau(P)$.

On dénote par $I\tau$ l'interprétation qui associe pour toute proposition $P \in \mathcal{P}_0$ la valeur de vérité $P^{I\tau} \stackrel{\text{def}}{=} \llbracket \tau(P) \rrbracket^I$; alors le lemme 6.7 de substitution propositionnelle dit que $\llbracket \varphi\tau \rrbracket^I = \llbracket \varphi \rrbracket^{I\tau}$. Cela implique en particulier que si φ est valide, alors $\varphi\tau$ l'est aussi, et permet de démontrer de nombreuses *équivalences usuelles*.

Le *théorème de compacité* énonce que, si un ensemble S de formules propositionnelles est insatisfiable, alors il existe un sous-ensemble fini $F \subseteq_{\text{fin}} S$ qui est insatisfiable. Par conséquent, si $S \models \varphi$, alors il existe $F \subseteq_{\text{fin}} S$ tel que $F \models \varphi$ (corollaire 6.14).

Comme nous l'avons vu dans l'exemple 5.5, il peut y avoir plusieurs formules propositionnelles avec la même sémantique. Le but de cette section est de mieux comprendre ce phénomène.

6.1. Conséquences logiques. Si S est un ensemble de formules propositionnelles et φ est une formule propositionnelle, on dit que φ est une *conséquence logique* de S et on écrit $S \models \varphi$ si pour toute interprétation I telle que $I \models S$ on a $I \models \varphi$ – autrement dit, si pour toute interprétation I , $I \models S$ implique $I \models \varphi$. ■ (DUPARC, 2015, sec. I.2.5)

Propriété 6.1. Soit φ une formule propositionnelle. Alors φ est valide si et seulement si $\emptyset \models \varphi$, c'est-à-dire φ est une conséquence logique de l'ensemble vide.

Démonstration. On a $\emptyset \models \varphi$

- si et seulement si, pour toute interprétation I , si I satisfait toutes les formules de l'ensemble vide, alors $I \models \varphi$,
- si et seulement si, pour toute interprétation I , on a $I \models \varphi$,
- si et seulement si, φ est valide. □

Dans le cas d'un ensemble $S = \{\psi\}$ constitué d'une seule formule propositionnelle ψ , on notera plus simplement $\psi \models \varphi$ et on dira que φ est une *conséquence logique* de ψ . Si on écrit $\text{Sat}(\varphi) \stackrel{\text{def}}{=} \{I \in \mathbb{B}^{\mathcal{P}_0} \mid I \models \varphi\}$ pour l'ensemble des interprétations qui satisfont une formule propositionnelle φ , $\psi \models \varphi$ revient à $\text{Sat}(\psi) \subseteq \text{Sat}(\varphi)$. On peut relier cette notion à la validité d'une seule formule propositionnelle comme suit.

Lemme 6.2 (déduction). *Soit S un ensemble de formules propositionnelles, et φ et ψ deux formules propositionnelles. Alors $S \cup \{\psi\} \models \varphi$ si et seulement si $S \models \psi \rightarrow \varphi$. En particulier quand $S = \emptyset$, $\psi \models \varphi$ si et seulement si $\psi \rightarrow \varphi$ est valide.*

Démonstration. On a $S \cup \{\psi\} \models \varphi$

- si et seulement si, pour toute interprétation I , si I satisfait toutes les formules propositionnelles de $S \cup \{\psi\}$, alors $I \models \varphi$,
- si et seulement si, pour toute interprétation I , si I satisfait toutes les formules propositionnelles de S , et si de plus $I \models \psi$, alors $I \models \varphi$,
- si et seulement si, pour toute interprétation I , si I satisfait toutes les formules propositionnelles de S , alors $I \models \psi \rightarrow \varphi$,
- si et seulement si, $S \models \psi \rightarrow \varphi$.

Par suite, le cas où $S = \emptyset$ découle de la propriété 6.1. □

Une autre façon de comprendre les conséquences logiques est de définir un *pré-ordre* à l'aide des sémantiques fonctionnelles $\llbracket \varphi \rrbracket$. On considère pour cela l'ordre $\mathbf{F} < \mathbf{V}$ sur les valeurs de vérités, et on dit que ψ est *fonctionnellement plus petite* que φ , noté $\llbracket \psi \rrbracket \leq \llbracket \varphi \rrbracket$, si pour toute interprétation I , $\llbracket \psi \rrbracket^I \leq \llbracket \varphi \rrbracket^I$. D'après cette définition, deux formules propositionnelles φ et ψ sont *fonctionnellement équivalentes*, c'est-à-dire telles que $\llbracket \varphi \rrbracket = \llbracket \psi \rrbracket$, si et seulement si $\llbracket \varphi \rrbracket \leq \llbracket \psi \rrbracket$ et $\llbracket \psi \rrbracket \leq \llbracket \varphi \rrbracket$.

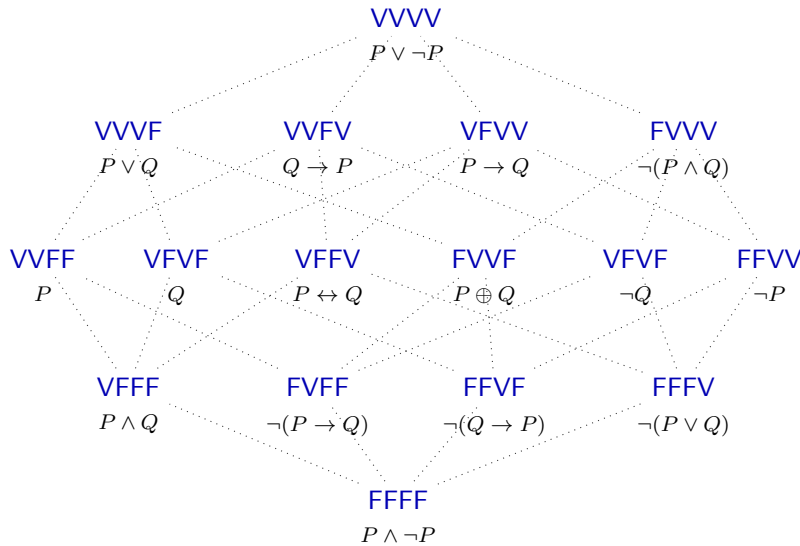
On peut visualiser le pré-ordre fonctionnel sous la forme d'un treillis comme celui de la figure 6.1. Dans cette figure, on donne pour chacun des 16 cas possibles sur deux propositions P et Q les valeurs de vérité des formules propositionnelles pour les interprétations partielles $[\mathbf{V}/P, \mathbf{V}/Q]$, $[\mathbf{V}/P, \mathbf{F}/Q]$, $[\mathbf{F}/P, \mathbf{V}/Q]$ et $[\mathbf{F}/P, \mathbf{F}/Q]$, dans cet ordre. Chaque élément du treillis est illustré par un exemple de formule propositionnelle avec cette sémantique ; il y en a bien sûr d'autres, comme $P \wedge \neg Q$ pour \mathbf{FVFF} ou $\neg P \vee \neg Q$ pour \mathbf{FVVV} , ou comme vu dans l'exemple 5.5, $(P \vee \neg Q) \wedge P$ pour \mathbf{VVFF} . Dans ce treillis, $\llbracket \psi \rrbracket \leq \llbracket \varphi \rrbracket$ s'il existe un chemin pointillé qui monte de $\llbracket \psi \rrbracket$ à $\llbracket \varphi \rrbracket$.

Lemme 6.3 (pré-ordre fonctionnel). *Soient φ et ψ deux formules propositionnelles. Alors $\psi \models \varphi$ si et seulement si $\llbracket \psi \rrbracket \leq \llbracket \varphi \rrbracket$.*

Démonstration. On a $\psi \models \varphi$

- si et seulement si, pour toute interprétation I , si $I \models \psi$ alors $I \models \varphi$,
- si et seulement si, pour toute interprétation I , si $\llbracket \psi \rrbracket^I = \mathbf{V}$ alors $\llbracket \varphi \rrbracket^I = \mathbf{V}$,
- si et seulement si, pour toute interprétation I , $\llbracket \psi \rrbracket^I \leq \llbracket \varphi \rrbracket^I$,
- si et seulement si, $\llbracket \psi \rrbracket \leq \llbracket \varphi \rrbracket$. □

6.2. Équivalences logiques. On dit que deux formules propositionnelles φ et ψ sont *logiquement équivalentes* si $\psi \models \varphi$ et $\varphi \models \psi$, c'est-à-dire si, pour toute interprétation I , $I \models \psi$ si et seulement si $I \models \varphi$. En terme d'ensemble de modèles, cela revient à demander $\text{Sat}(\psi) = \text{Sat}(\varphi)$. Par le lemme 6.2 de déduction, cela se produit si et seulement si $\psi \leftrightarrow \varphi$

FIGURE 6.1. Le treillis du pré-ordre fonctionnel sur deux propositions P et Q .

est une formule propositionnelle valide. Par le lemme 6.3 de pré-ordre fonctionnel, cela se produit si et seulement si φ et ψ sont fonctionnellement équivalentes, c'est-à-dire si et seulement si $\llbracket \varphi \rrbracket = \llbracket \psi \rrbracket$.

Comme nous allons le voir, l'équivalence logique permet de remplacer une formule propositionnelle, qui représente par exemple une expression booléenne d'un programme ou un circuit logique, par une autre formule propositionnelle équivalente potentiellement plus efficace à évaluer; ainsi, dans l'exemple 5.5, la formule propositionnelle P est plus facile à évaluer que $(P \vee \neg Q) \wedge P$.

Il est donc très utile de savoir dire si deux formules propositionnelles φ et ψ sont logiquement équivalentes ou non. Cela peut se faire à l'aide de tables de vérité

- d'après le lemme 6.2 de déduction, en vérifiant si $\varphi \leftrightarrow \psi$ est valide, ou
- d'après le lemme 6.3 de pré-ordre fonctionnel, en vérifiant si $\llbracket \varphi \rrbracket = \llbracket \psi \rrbracket$.

Exemple 6.4. Appliquons les lemmes 6.2 et 6.3 à l'équivalence entre $\neg(P \vee Q)$ et $\neg P \wedge \neg Q$. La table de vérité correspondante est donnée dans la table 6.1 ci-dessous.

On vérifie dans cette table que la troisième colonne ($\neg(P \vee Q)$) et la cinquième colonne ($\neg P \wedge \neg Q$) sont identiques, donc ces deux formules propositionnelles sont logiquement équivalentes par le lemme 6.3 de pré-ordre fonctionnel. La sixième colonne ($\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$) ne contient que des **V**, donc cette formule propositionnelle est valide, ce qui montre aussi que les deux formules propositionnelles sont logiquement équivalentes par le lemme 6.2 de déduction.

Cette approche via les tables de vérité a l'inconvénient de nécessiter de tester toutes les interprétations des propositions de $\text{Props}(\varphi) \cup \text{Props}(\psi)$, soit un nombre exponentiel de possibilités. Nous verrons des techniques qui permettent souvent de n'explorer qu'une toute

TABLE 6.1. Table de vérité de $\neg(P \vee Q)$, $\neg P \wedge \neg Q$ et $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$.

P	Q	$P \vee Q$	$\neg(P \vee Q)$	$\neg P$	$\neg Q$	$\neg P \wedge \neg Q$	$\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
V	V	V	F	F	F	F	V
V	F	V	F	F	V	F	V
F	V	V	F	V	F	F	V
F	F	F	V	V	V	V	V

petite partie de cet espace de recherche. En attendant, nous allons voir une approche qui s'applique bien à de petites formules propositionnelles et des raisonnements « à la main ».

■ (DUPARC, 2015, sec. I.2.6),
(GOUBAULT-LARRECQ et MACKIE, 1997, def. 2.4), (DAVID, NOUR et RAFFALLI, 2003, def. 4.6.4),
(HARRISSON, 2009, p. 41)

6.3. Substitutions propositionnelles. Une *substitution propositionnelle* (aussi appelée une

« transduction propositionnelle ») est une fonction τ qui associe à chaque proposition $P \in \mathcal{P}_0$ une formule propositionnelle $\tau(P)$, de tel sorte que son *domaine* $\text{dom}(\tau) \stackrel{\text{def}}{=} \{P \in \mathcal{P}_0 \mid \tau(P) \neq P\}$ soit fini. On écrit $[\varphi_1/P_1, \dots, \varphi_n/P_n]$ pour la substitution propositionnelle de domaine $\{P_1, \dots, P_n\}$ où les P_i sont distinctes et qui associe φ_i à P_i . Toute substitution propositionnelle se relève en une fonction des formules propositionnelles dans les formules propositionnelles :

$$P\tau \stackrel{\text{def}}{=} \tau(P), \quad (\neg\varphi)\tau \stackrel{\text{def}}{=} \neg(\varphi\tau), \quad (\varphi \vee \psi)\tau \stackrel{\text{def}}{=} (\varphi\tau) \vee (\psi\tau), \quad (\varphi \wedge \psi)\tau \stackrel{\text{def}}{=} (\varphi\tau) \wedge (\psi\tau).$$

Exemple 6.5. Considérons la formule propositionnelle $\psi = (P \wedge Q)$, ainsi que la substitution propositionnelle $\tau = [(P \vee \neg Q)/P, P/Q]$. Alors $\psi\tau = (P \vee \neg Q) \wedge P$.

6.3.1. Lemme de substitution propositionnelle. Pour une interprétation I et une substitution propositionnelle τ , on définit l'interprétation $I\tau$ comme associant $P^{I\tau} \stackrel{\text{def}}{=} \llbracket \tau(P) \rrbracket^I$ à chaque proposition P de \mathcal{P}_0 .

Exemple 6.6. Soit I une interprétation qui étend $[F/P, F/Q]$ et $\tau = [(P \vee \neg Q)/P, P/Q]$. Alors $P^{I\tau} = \llbracket P \vee \neg Q \rrbracket^I = \mathbf{V}$ et $Q^{I\tau} = \llbracket P \rrbracket^I = \mathbf{F}$.

■ (GOUBAULT-LARRECQ et MACKIE, 1997, thm. 2.10),
(HARRISSON, 2009, thm. 2.3)

Lemme 6.7 (substitution propositionnelle). *Pour toute formule propositionnelle φ , toute substitution propositionnelle τ et toute interprétation I , $\llbracket \varphi\tau \rrbracket^I = \llbracket \varphi \rrbracket^{I\tau}$.*

Démonstration. Par induction structurelle sur φ . Pour le cas de base où $\varphi = P \in \mathcal{P}_0$,

$$\llbracket P\tau \rrbracket^I = \llbracket \tau(P) \rrbracket^I = \llbracket P \rrbracket^{I\tau}.$$

Pour l'étape d'induction où $\varphi = \neg\psi$,

$$\llbracket (\neg\psi)\tau \rrbracket^I = \llbracket \neg(\psi\tau) \rrbracket^I = \text{non } \llbracket \psi\tau \rrbracket^I \stackrel{\text{h.i.}}{=} \text{non } \llbracket \psi \rrbracket^{I\tau} = \llbracket \neg\psi \rrbracket^{I\tau}.$$

Pour l'étape d'induction où $\varphi = \varphi' \vee \psi$,

$$\llbracket (\varphi' \vee \psi)\tau \rrbracket^I = \llbracket (\varphi'\tau) \vee (\psi\tau) \rrbracket^I = \llbracket \varphi'\tau \rrbracket^I \text{ ou } \llbracket \psi\tau \rrbracket^I \stackrel{\text{h.i.}}{=} \llbracket \varphi' \rrbracket^{I\tau} \text{ ou } \llbracket \psi \rrbracket^{I\tau} = \llbracket \varphi' \vee \psi \rrbracket^{I\tau}.$$

L'étape d'induction où $\varphi = \varphi' \wedge \psi$ est similaire. □

Exemple 6.8. Considérons comme dans les exemples 6.5 et 6.6 une interprétation I qui étend $[F/P, F/Q]$, la substitution propositionnelle $\tau = [(P \vee \neg Q)/P, P/Q]$, et la formule propositionnelle $\psi = (P \wedge Q)$. Alors d'un côté $\llbracket \psi \tau \rrbracket^I = \llbracket (P \vee \neg Q) \wedge P \rrbracket^{[F/P, F/Q]} = F$, et de l'autre $\llbracket \psi \rrbracket^{I\tau} = \llbracket P \wedge Q \rrbracket^{[V/P, F/Q]} = F$.

Corollaire 6.9. Soit φ une formule propositionnelle valide et τ une substitution propositionnelle. Alors $\varphi\tau$ est valide. ■ (HARRISSON, 2009, cor. 2.4)

Démonstration. Par le lemme 6.7 de substitution propositionnelle, pour toute interprétation I , $\llbracket \varphi\tau \rrbracket^I = \llbracket \varphi \rrbracket^{I\tau}$. Or, comme φ est valide, $\llbracket \varphi \rrbracket^{I\tau} = V$. □

Exemple 6.10. Voyons tout de suite une application de la corollaire 6.9. Nous avons vu dans l'exemple 6.4 que $\neg(P \vee Q) \leftrightarrow (\neg P \wedge \neg Q)$ est valide. Dès lors, pour toutes formules propositionnelles φ et ψ , on peut appliquer la substitution propositionnelle $[\varphi/P, \psi/Q]$ et déduire que $\neg(\varphi \vee \psi) \leftrightarrow (\neg\varphi \wedge \neg\psi)$ est valide – autrement dit, que $\neg(\varphi \vee \psi)$ et $\neg\varphi \wedge \neg\psi$ sont logiquement équivalentes.

6.4. Équivalences usuelles. En appliquant le même raisonnement que dans l'exemple 6.10, ■ (HARRISSON, 2009, pp. 44–46) on a plus généralement les équivalences logiques suivantes.

Résumé. Pour toutes formules propositionnelles φ , ψ , et ψ' ,

$(\varphi \vee \varphi) \leftrightarrow \varphi$,	(idempotence de \vee)
$(\varphi \vee \psi) \leftrightarrow (\psi \vee \varphi)$,	(commutativité de \vee)
$((\varphi \vee \psi) \vee \psi') \leftrightarrow (\varphi \vee (\psi \vee \psi'))$	(associativité de \vee)
$(\varphi \wedge \varphi) \leftrightarrow \varphi$,	(idempotence de \wedge)
$(\varphi \wedge \psi) \leftrightarrow (\psi \wedge \varphi)$,	(commutativité de \wedge)
$((\varphi \wedge \psi) \wedge \psi') \leftrightarrow (\varphi \wedge (\psi \wedge \psi'))$	(associativité de \wedge)
$\neg\neg\varphi \leftrightarrow \varphi$,	(double négation)
$(\varphi \wedge (\psi \vee \psi')) \leftrightarrow ((\varphi \wedge \psi) \vee (\varphi \wedge \psi'))$,	(distributivité de \wedge sur \vee)
$(\varphi \vee (\psi \wedge \psi')) \leftrightarrow ((\varphi \vee \psi) \wedge (\varphi \vee \psi'))$,	(distributivité de \vee sur \wedge)
$\neg(\varphi \vee \psi) \leftrightarrow (\neg\varphi \wedge \neg\psi)$,	(dualité de DE MORGAN pour \vee)
$\neg(\varphi \wedge \psi) \leftrightarrow (\neg\varphi \vee \neg\psi)$,	(dualité de DE MORGAN pour \wedge)
$(\varphi \rightarrow \psi) \leftrightarrow (\neg\varphi \vee \psi)$,	(définition de l'implication)
$(\varphi \rightarrow \psi) \leftrightarrow (\neg\psi \rightarrow \neg\varphi)$,	(contraposition)
$((\varphi \wedge \psi) \rightarrow \psi') \leftrightarrow (\varphi \rightarrow (\psi \rightarrow \psi'))$.	(curryfication)

Ces équivalences logiques sont aussi très utiles pour simplifier des formules propositionnelles. On repose pour cela sur la propriété suivante.

Corollaire 6.11. Soit φ une formule propositionnelle, et τ et τ' deux substitutions propositionnelles telles que, pour toute proposition $P \in \mathcal{P}_0$, $\tau(P)$ soit logiquement équivalente à $\tau'(P)$. Alors $\varphi\tau$ est logiquement équivalente à $\varphi\tau'$.

Démonstration. Pour montrer que $\varphi\tau$ et $\varphi\tau'$ sont logiquement équivalentes, on va montrer que pour toute interprétation $I \in \mathbb{B}^{\mathcal{P}_0}$, $\llbracket \varphi\tau \rrbracket^I = \llbracket \varphi\tau' \rrbracket^I$.

Par hypothèse, pour toute proposition $P \in \mathcal{P}_0$, $\tau(P)$ et $\tau'(P)$ sont logiquement équivalentes. Cela signifie que pour toute interprétation I et toute proposition P , par définition de $I\tau$ et $I\tau'$, $P^{I\tau} = \llbracket \tau(P) \rrbracket^I = \llbracket \tau'(P) \rrbracket^I = P^{I\tau'}$, c'est-à-dire que pour toute interprétation I , $I\tau = I\tau'$ sont une seule et même interprétation. On en déduit par le lemme 6.7 de substitution propositionnelle que $\llbracket \varphi\tau \rrbracket^I = \llbracket \varphi \rrbracket^{I\tau} = \llbracket \varphi\tau' \rrbracket^I = \llbracket \varphi \rrbracket^{I\tau'}$. \square

Exemple 6.12. Voici une illustration : on souhaite montrer que $(P \rightarrow Q) \rightarrow P$ est logiquement équivalente à P . Les deux substitutions propositionnelles $[(P \rightarrow Q)/P, P/Q]$ et $[(\neg P \vee Q)/P, P/Q]$ sont bien telles que $(P \rightarrow Q) \leftrightarrow (\neg P \vee Q)$ (par définition de l'implication) et $P \leftrightarrow P$ est immédiat. En appliquant la corollaire 6.11 à la formule propositionnelle $\varphi = (P \rightarrow Q)$, on en déduit que $(P \rightarrow Q) \rightarrow P$ est logiquement équivalente à $(\neg P \vee Q) \rightarrow P$.

Puis, par des raisonnements similaires, par définition de l'implication, elle est logiquement équivalente à $\neg(\neg P \vee Q) \vee P$, puis par dualité de DE MORGAN pour \vee , à $(\neg\neg P \wedge \neg Q) \vee P$, par double négation, à $(P \wedge \neg Q) \vee P$, par commutativité de \vee , à $P \vee (P \wedge \neg Q)$, par distributivité de \vee sur \wedge , à $(P \vee P) \wedge (P \vee \neg Q)$, par idempotence de \vee , à $P \wedge (P \vee \neg Q)$, par commutativité de \wedge , à $(P \vee \neg Q) \wedge P$, qui comme nous l'avons vu dans l'exemple 5.5 est logiquement équivalente à P .

6.5. Compacité. Le théorème de compacité énonce qu'un ensemble S de formules propositionnelles est satisfiable si et seulement si tous ses sous-ensembles finis le sont aussi ; sous forme contrapositive, $S \models \perp$ si et seulement si $\exists F \subseteq_{\text{fin}} S$ tel que $F \models \perp$.

Théorème 6.13 (compacité). *Soit S un ensemble insatisfiable de formules propositionnelles. Alors il existe un sous-ensemble fini F de S qui est déjà insatisfiable.*

Ce théorème peut être montré

- comme une conséquence du théorème de TYCHONOFF, voir la section 6.5.1 ;
- par la construction d'arbres sémantiques et le lemme de KÖNIG, voir la section 6.5.2 ;
- comme une conséquence de la correction et complétude de systèmes de preuves.

Voici enfin une formulation équivalente du théorème en termes de conséquences logiques.

Corollaire 6.14. *Soit S un ensemble de formules propositionnelles et φ une formule propositionnelle. Si $S \models \varphi$, alors il existe un sous-ensemble fini F de S tel que $F \models \varphi$.*

Démonstration. Si $S \models \varphi$, alors $S \cup \{\neg\varphi\}$ est insatisfiable. Par le théorème de compacité, il existe un sous-ensemble fini $F' \subseteq_{\text{fin}} S \cup \{\neg\varphi\}$ qui est déjà insatisfiable. Soit $F \stackrel{\text{def}}{=} F' \setminus \{\neg\varphi\}$. Par contraposée, pour toute interprétation I , si $I \not\models \varphi$ alors $I \not\models F$, sans quoi on aurait $I \models F'$ qui contredirait son insatisfiabilité. \square

6.5.1. Compacité par le théorème de TYCHONOFF. Cette preuve est très simple mais nécessite d'introduire quelques notions de base en topologie. À noter que cette preuve ne nécessite pas de supposer \mathcal{P}_0 dénombrable, puisque le théorème de TYCHONOFF s'applique aussi dans le cas non dénombrable.

Espaces topologiques. Un *espace topologique* est un ensemble T muni d'une *topologie* \mathcal{O} , qui est une collection non vide de sous-ensembles dits *ouverts*, telle que toute intersection finie d'ouverts soit elle-même un ouvert, et toute union arbitraire d'ouverts soit elle-même un ouvert. À noter que $T = \bigcap_{\emptyset} T$ et $\emptyset = \bigcup_{\emptyset} \emptyset$ sont ouverts. Les ensembles *fermés* sont définis comme les compléments $T \setminus O$ d'ouverts O de \mathcal{O} ; ils sont fermés par unions finies et intersections arbitraires.

La *topologie discrète* sur T est celle où les ouverts sont exactement les sous-ensembles de T . Cette topologie a plusieurs propriétés.

- Si O est un ouvert, alors c'est aussi un fermé et inversement.
- Un espace topologique T est *séparé* si pour tous $x \neq x' \in T$, il existe deux ouverts disjoints O et O' tels que $x \in O$ et $x' \in O'$. Un espace topologique discret est nécessairement séparé puisqu'il suffit de poser $O \stackrel{\text{def}}{=} \{x\}$ et $O' \stackrel{\text{def}}{=} \{x'\}$.

Un *recouvrement* d'un espace topologique T est une famille \mathcal{C} d'ouverts telle que $\bigcup_{O \in \mathcal{C}} O = T$. L'espace T est *compact* s'il est séparé et que de tout recouvrement on peut extraire un recouvrement fini. Si T est fini et séparé, alors il est nécessairement compact.

Soit $(T_j)_{j \in J}$ une famille d'espaces topologiques indexés par un ensemble J . Alors le produit cartésien $\prod_{j \in J} T_j$ muni de la *topologie produit* est un espace topologique. Dans cette topologie, les ouverts sont des unions de produits de la forme $\prod_{j \in J} O_j$ où chaque O_j est un ouvert de T_j , mais où seul un nombre fini de tels O_j sont différents de l'espace T_j en entier. Il s'agit bien d'une topologie : les ouverts sont bien fermés par union par définition, et pour l'intersection finie, par distributivité on se ramène au cas $(\prod_{j \in J} O_j) \cap (\prod_{j \in J} O'_j) = \prod_{j \in J} (O_j \cap O'_j)$ où seul un nombre fini de $O_j \cap O'_j$ peut être différent de l'espace T_j en entier.

Théorème 6.15 (TYCHONOFF). *Soit $(T_j)_{j \in J}$ une famille d'espaces topologiques compacts. Alors $\prod_{j \in J} T_j$ est compact.*

Démonstration du théorème de compacité. L'ensemble $\mathbb{B} = \{\mathbf{F}, \mathbf{V}\}$ des valeurs de vérité muni de la topologie discrète est séparé, et comme il est fini, il est compact. Par le théorème de TYCHONOFF, $\mathbb{B}^{\mathcal{P}_0}$ muni de la topologie produit est compact; c'est l'espace des interprétations des variables propositionnelles de \mathcal{P}_0 .

On montre par induction sur les formules propositionnelles φ que l'ensemble $\text{Sat}(\varphi)$ des interprétations qui satisfont φ est un sous-ensemble à la fois ouvert et fermé pour la topologie produit sur $\mathbb{B}^{\mathcal{P}_0}$. Pour le cas de base, $\text{Sat}(P)$ est le produit $\{\mathbf{V}\} \times \prod_{P' \neq P} \mathbb{B}$ associant \mathbf{V} à P et $\mathbb{B} = \{\mathbf{F}, \mathbf{V}\}$ à toutes les propositions $P' \neq P$, qui est bien un ouvert (seule la proposition P est envoyée sur l'ouvert $\{\mathbf{V}\}$ de \mathbb{B} muni de la topologie discrète, les autres propositions sont envoyées sur l'espace \mathbb{B} en entier) fermé (c'est le complémentaire de $\{\mathbf{F}\} \times \prod_{P' \neq P} \mathbb{B}$ qui est aussi un ouvert de $\mathbb{B}^{\mathcal{P}_0}$). Pour l'étape d'induction, $\text{Sat}(\neg\varphi) = \mathbb{B}^{\mathcal{P}_0} \setminus \text{Sat}(\varphi)$ est bien un ouvert fermé et $\text{Sat}(\varphi \vee \psi) = \text{Sat}(\varphi) \cup \text{Sat}(\psi)$ est bien un ouvert fermé.

Étant donné S un ensemble insatisfiable de formules propositionnelles, toute interprétation I dans $\mathbb{B}^{\mathcal{P}_0}$ satisfait au moins une des formules $\neg\varphi$ pour $\varphi \in S$, donc $\mathbb{B}^{\mathcal{P}_0} = \bigcup_{\varphi \in S} \text{Sat}(\neg\varphi)$. Par compacité, $\mathbb{B}^{\mathcal{P}_0}$ a un recouvrement fini $\bigcup_{\varphi \in F} \text{Sat}(\neg\varphi)$; cet ensemble $F \subseteq S$ est fini et insatisfiable. \square

6.5.2. *Compacité par arbres sémantiques et le lemme de KÖNIG.* Soit S un ensemble de formules et $\text{Props}(S)$ l'ensemble des propositions qui apparaissent dans S . Un *arbre sémantique*

■ (GOUBAULT-LARRECQ et MACKIE, 1997, thm. 2.25)

■ (CHANG et LEE, 1973, sec. 4.4)

☞ Comme \mathcal{P}_0 est supposé dénombrable, $\text{Props}(S)$ l'est aussi. Une façon simple de construire un arbre sémantique est de fixer une énumération P_1, P_2, \dots sans répétition de $\text{Props}(S)$ et d'étiqueter tous les nœuds de profondeur n de l'arbre binaire complet de hauteur $|\text{Props}(S)|$ par la proposition P_n .

pour S est un arbre binaire, généralement infini si $\text{Props}(S)$ est infini, dont les nœuds sont étiquetés par des propositions de $\text{Props}(S)$ et tel que, pour toute branche de l'arbre, chaque proposition de $\text{Props}(S)$ étiquette exactement un nœud de la branche.

Étant donné un arbre sémantique pour S , on peut associer à chaque nœud une *interprétation partielle*. La racine de l'arbre est associée à l'interprétation de domaine vide. Pour tout nœud étiqueté par P et d'interprétation I , le fils gauche est associé à I étendue par $[F/P]$ et le fils droit à I étendue par $[V/P]$. Une branche de l'arbre sémantique est donc associée à une interprétation de domaine $\text{Props}(S)$ entier. Pour une interprétation partielle I et une formule φ , on écrit $I \models \varphi$ s'il existe une interprétation totale I' qui étend I (c'est-à-dire avec $P^I = P^{I'}$ pour tout $P \in \text{dom}(I)$) telle que $I' \models \varphi$. On appelle un *nœud d'échec* pour $\varphi \in S$ un nœud minimal (le plus proche de la racine) d'interprétation partielle I tel que $I \not\models \varphi$: quel que soit le choix d'interprétation des propositions P qui n'ont pas été fixées dans I , on est sûr de ne pas satisfaire S .

Supposons S insatisfiable. Pour toute interprétation I , il existe une formule $\varphi \in S$ telle que $I \not\models \varphi$. Donc toutes les branches d'un arbre sémantique pour S contiennent un nœud d'échec. Un *arbre sémantique fermé* de S est un arbre sémantique élagué dès que l'on rencontre un nœud d'échec ; on étiquette alors ce nœud d'échec par une *formule témoin* $\varphi \in S$ telle que $I \not\models \varphi$ où I est l'interprétation partielle du nœud. Par le lemme de KÖNIG, un arbre sémantique fermé est nécessairement fini. La figure 6.2 montre un arbre sémantique fermé pour l'exemple 5.12.

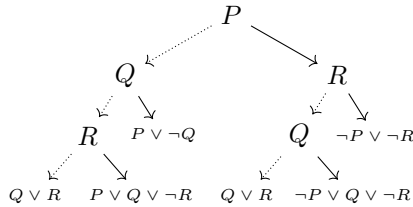


FIGURE 6.2. Arbre sémantique fermé pour l'ensemble de formules de l'exemple 5.12.

Démonstration du théorème de compacité. On considère un arbre sémantique fermé de S . Comme il est fini, il y a un ensemble fini $F \subseteq S$ de formules témoin. De plus, pour toute interprétation I , il existe un nœud d'échec le long de la branche correspondante de l'arbre sémantique et une formule témoin $\varphi \in F$ telle que $I \not\models \varphi$. Donc F est insatisfiable. \square

7. FORMES NORMALES

Résumé. On peut mettre n'importe quelle formule propositionnelle sous *forme normale négative* en « poussant » les négations vers les feuilles par application de la double négation et des dualités de DE MORGAN ; la formule propositionnelle obtenue est logiquement équivalente et de la forme

$$P \quad \text{ou} \quad \neg P \quad \text{ou} \quad \bigvee \begin{array}{l} \varphi \\ \psi \end{array} \quad \text{ou} \quad \bigwedge \begin{array}{l} \varphi \\ \psi \end{array}$$

où $P \in \mathcal{P}_0$ et φ et ψ sont des formules propositionnelles sous forme normale négative. Les formules propositionnelles de la forme « P » ou « $\neg P$ » sont appelées des *littéraux*. On note « $\bar{\varphi}$ » pour la forme normale négative de $\neg\varphi$.

Une formule propositionnelle est sous *forme normale conjonctive* si elle s'écrit comme

$$\bigwedge_{1 \leq i \leq m} \bigvee_{1 \leq j \leq n_i} l_{i,j}$$

où les $l_{i,j}$ sont des littéraux ; on appelle chaque disjonction $\bigvee_{1 \leq j \leq n_i} l_{i,j}$ une *clause*. On peut mettre une formule propositionnelle sous forme normale conjonctive à partir d'une formule propositionnelle sous forme normale négative en « poussant » les disjonctions vers le bas par application de la distributivité de \bigvee sur \bigwedge . Une formule propositionnelle est sous *forme normale disjonctive* si elle s'écrit comme

$$\bigvee_{1 \leq i \leq m} \bigwedge_{1 \leq j \leq n_i} l_{i,j}$$

où les $l_{i,j}$ sont des littéraux ; on appelle chaque conjonction $\bigwedge_{1 \leq j \leq n_i} l_{i,j}$ satisfiable un *monôme*. On peut mettre une formule propositionnelle sous forme normale disjonctive à partir d'une formule propositionnelle sous forme normale négative en « poussant » les conjonctions vers le bas par application de la distributivité de \bigwedge sur \bigvee . Les formules propositionnelles sous forme normale conjonctive ou disjonctive obtenues ainsi sont logiquement équivalentes à la formule d'origine mais potentiellement de taille exponentielle (exemples 7.5 et 7.22).

Dans le cas des formes normales conjonctives, on préfère en pratique construire des *équi-satisfiables* avec la formule d'origine (section 7.2.2) ; cette opération a un coût linéaire dans le pire des cas.

Dans le cas des formes normales disjonctives, on peut chercher à mettre la formule sous forme *première*, qui est une forme canonique (section 7.3.2). Cette forme première peut ensuite être *minimisée* (section 7.3.3).

Un dernier exemple de forme normale est l'emploi de *diagrammes binaires de décision* (section 7.4), qui dans leur version ordonnée et réduite donnent aussi lieu à une forme canonique (théorème 7.25).

Nous avons vu dans la section précédente qu'il existe de nombreuses manières d'écrire des formules propositionnelles logiquement équivalentes (c.f. section 6). Parmi toutes ces formules propositionnelles équivalentes, certaines seront plus faciles à traiter ; par exemple, la formule propositionnelle P est plus facile à évaluer que la formule propositionnelle $(P \vee \neg Q) \wedge P$. En particulier, les algorithmes de recherche de modèle ou de recherche de preuve que nous verrons par la suite travaillent sur des formules propositionnelles avec une syntaxe restreinte – on parle alors de *forme normale*.

■ (DAVID, NOUR et RAFFALI, 2003, sec. 2.6),
(GOUBAULT-LARRECQ et MACKIE, 1997, def. 2.38),
(HARRISSON, 2009, sec. 2.5)

7.1. Forme normale négative. Une formule propositionnelle est sous *forme normale négative* si elle respecte la syntaxe abstraite

$$\begin{aligned} \ell &::= P \mid \neg P && \text{(littéraux)} \\ \varphi &::= \ell \mid \varphi \vee \psi \mid \varphi \wedge \psi && \text{(formules propositionnelles sous forme normale négative)} \end{aligned}$$

où P est une proposition de \mathcal{P}_0 . En d'autres termes, les négations ne peuvent apparaître que devant des formules atomiques. Par exemple, la formule propositionnelle $\varphi_{\text{ex}} = (P \vee \neg Q) \wedge P$ de la figure 4.1 est sous forme normale négative.

La mise sous forme normale négative procède en « poussant » les négations dans l'arbre de syntaxe abstraite de la formule propositionnelle vers les feuilles.

Définition 7.1 (forme normale négative). Pour une formule propositionnelle φ , on notera $\text{nnf}(\varphi)$ sa forme normale négative obtenue inductivement par

$$\begin{aligned} \text{nnf}(P) &\stackrel{\text{def}}{=} P, & \text{nnf}(\neg P) &\stackrel{\text{def}}{=} \neg P, \\ \text{nnf}(\varphi \vee \psi) &\stackrel{\text{def}}{=} \text{nnf}(\varphi) \vee \text{nnf}(\psi), & \text{nnf}(\neg(\varphi \vee \psi)) &\stackrel{\text{def}}{=} \text{nnf}(\neg\varphi) \wedge \text{nnf}(\neg\psi), \\ \text{nnf}(\varphi \wedge \psi) &\stackrel{\text{def}}{=} \text{nnf}(\varphi) \wedge \text{nnf}(\psi), & \text{nnf}(\neg(\varphi \wedge \psi)) &\stackrel{\text{def}}{=} \text{nnf}(\neg\varphi) \vee \text{nnf}(\neg\psi), \\ & & \text{nnf}(\neg\neg\varphi) &\stackrel{\text{def}}{=} \text{nnf}(\varphi). \end{aligned}$$

On notera aussi en général

$$\overline{\varphi} \stackrel{\text{def}}{=} \text{nnf}(\neg\varphi)$$

pour la forme normale négative de la négation de φ , appelée la *formule duale* de φ .

À noter que les définitions dans la colonne de gauche de la définition 7.1 sont celles pour des formules propositionnelles qui n'ont pas de symbole « \neg » à leur racine, tandis que celles de la colonne de droite s'occupent des différents cas de formules propositionnelles enracinées par « \neg ». En termes algorithmiques, cette mise sous forme normale négative se fait en temps linéaire. Par les deux lois de dualité de DE MORGAN pour \vee et pour \wedge et par la loi de double négation, la mise sous forme normale négative préserve la sémantique des formules propositionnelles : φ et $\text{nnf}(\varphi)$ sont équivalentes.

Exemple 7.2. La loi de PEIRCE $((P \rightarrow Q) \rightarrow P) \rightarrow P$ s'écrit $\neg(\neg(\neg P \vee Q) \vee P) \vee P$ en syntaxe non étendue. Sa forme normale négative est

$$\begin{aligned} \text{nnf}(\neg(\neg(\neg P \vee Q) \vee P) \vee P) &= \text{nnf}(\neg(\neg(\neg P \vee Q) \vee P)) \vee P \\ &= (\text{nnf}(\neg\neg(\neg P \vee Q)) \wedge \neg P) \vee P \\ &= ((\neg P \vee Q) \wedge \neg P) \vee P. \end{aligned}$$

Sa formule duale est

$$\begin{aligned} \overline{\neg(\neg(\neg P \vee Q) \vee P)} &= \text{nnf}(\neg(\neg(\neg(\neg P \vee Q) \vee P) \vee P)) \\ &= \text{nnf}(\neg\neg(\neg(\neg P \vee Q) \vee P)) \wedge \neg P \\ &= \text{nnf}(\neg(\neg P \vee Q) \vee P) \wedge \neg P \\ &= (\text{nnf}(\neg(\neg P \vee Q)) \vee P) \wedge \neg P \\ &= ((\text{nnf}(\neg\neg P) \wedge \neg Q) \vee P) \wedge \neg P \\ &= ((P \wedge \neg Q) \vee P) \wedge \neg P. \end{aligned}$$

7.2. Forme clausale. Une formule propositionnelle en forme normale négative n'a que des opérateurs \vee et \wedge en guise de nœuds internes, sauf potentiellement des \neg juste au-dessus des propositions. En utilisant les lois de distributivité, on peut encore normaliser ces formules propositionnelles pour imposer que tous les \wedge soient au-dessus des \vee (forme normale conjonctive) ou vice-versa (forme normale disjonctive). La forme clausale est ensuite simplement une écriture « ensembliste » d'une formule propositionnelle sous forme normale conjonctive, et est largement employée dans les algorithmes de recherche de modèle et de recherche de preuve.

Nous allons voir deux techniques pour mettre une formule propositionnelle sous forme normale conjonctive. La première est très simple et s'appuie sur la loi de distributivité de \vee sur \wedge et construit une formule propositionnelle logiquement équivalente. Cependant, elle peut avoir un coût prohibitif en pratique, et nous verrons ensuite une technique qui construit une formule propositionnelle *équi-satisfiable*, au sens suivant :

Définition 7.3 (équivalent satisfiable). Soit φ et ψ deux formules propositionnelles. Elles sont *équivalent satisfiables* si φ est satisfiable si et seulement si ψ est satisfiable – autrement dit, $\exists I. I \models \varphi$ si et seulement si $\exists I'. I' \models \psi$.

7.2.1. Forme clausale logiquement équivalente. Soit φ une formule propositionnelle en forme normale négative. En utilisant de manière répétée la loi de distributivité de \vee sur \wedge , on « pousse » les disjonctions vers le bas et on obtient une mise sous *forme normale conjonctive* $\text{cnf}(\varphi)$ pour toute φ en forme normale négative. Le cas le plus important de cette transformation est le suivant :

$$\text{cnf}(\varphi \vee (\psi \wedge \psi')) \stackrel{\text{def}}{=} \text{cnf}((\psi \wedge \psi') \vee \varphi) \stackrel{\text{def}}{=} \text{cnf}(\varphi \vee \psi) \wedge \text{cnf}(\varphi \vee \psi').$$

À noter que la formule φ est *dupliquée* lors de cette transformation. Cela explique que la mise sous forme normale conjonctive peut avoir un coût exponentiel (voir l'exemple 7.5 ci-dessous). Une formule propositionnelle sous forme normale conjonctive s'écrit donc sous la forme

$$\bigwedge_{1 \leq i \leq m} \bigvee_{1 \leq j \leq n_i} l_{i,j}$$

■ (DUPARC, 2015, sec. I.2.10.2),
(DAVID, NOUR et RAFFALLI, 2003,
sec. 7.4.2)

où les $\ell_{i,j}$ sont des littéraux. Les sous-formules $C_i \stackrel{\text{def}}{=} \bigvee_{1 \leq j \leq n_i} \ell_{i,j}$ sont appelées les *clauses* de la formule.

k-CNF. Quand les clauses sont des disjonctions d'au plus *k* littéraux, on dit que la formule est sous forme « *k*-CNF ». Par exemple, la formule propositionnelle $\varphi_{\text{ex}} = (P \vee \neg Q) \wedge P$ de la figure 4.1 est déjà sous forme normale conjonctive : $\text{cnf}(\varphi_{\text{ex}}) = \varphi_{\text{ex}}$. Elle est composée de deux clauses : $P \vee \neg Q$ et P ; comme ces deux clauses contiennent chacune au plus deux littéraux, c'est une formule en 2-CNF.

Forme clause, forme k-clause. Par les lois d'idempotence de \vee , de commutativité de \vee et d'associativité de \vee , chaque clause peut-être vue comme un ensemble de littéraux, pour lequel les notations ensemblistes \in et \subseteq s'appliquent; par exemple, la clause $P \vee Q \vee \neg R$ peut être vue comme l'ensemble de littéraux $\{P, Q, \neg R\}$. De même, par les lois d'idempotence de \wedge , de commutativité de \wedge et d'associativité de \wedge , on peut voir une formule propositionnelle en forme normale conjonctive comme un ensemble de clauses.

Pour une formule propositionnelle φ donnée, on note $\text{Cl}(\varphi)$ l'ensemble des clauses de $\text{cnf}(\varphi)$ (où chaque clause est vue comme un ensemble); on appelle cela sa *forme clause*. Pour la formule propositionnelle $\varphi_{\text{ex}} = (P \vee \neg Q) \wedge P$ de la figure 4.1, $\text{Cl}(\varphi_{\text{ex}}) = \{\{P, \neg Q\}, \{P\}\}$. L'ensemble de formules de l'exemple 5.12 peut aussi être vu comme une forme clause

$$\{\{P, Q, \neg R\}, \{Q, R\}, \{\neg P, \neg Q, R\}, \{\neg P, \neg R\}, \{P, \neg Q\}\}.$$

Quand les clauses contiennent au plus *k* littéraux, on parle aussi de forme *k-clause*.

Exemple 7.4. Comme vu dans l'exemple 7.2, la formule duale de la loi de PEIRCE $((P \rightarrow Q) \rightarrow P) \wedge \neg P$ s'écrit $((P \wedge \neg Q) \vee P) \wedge \neg P$. La mise sous forme conjonctive produit $(P \vee P) \wedge (\neg Q \vee P) \wedge \neg P$ et donc la forme clause $\{\{P, P\}, \{\neg Q, P\}, \{\neg P\}\}$.

Exemple 7.5. La mise sous forme normale conjonctive peut avoir un coût exponentiel du fait des duplications de formules. Par exemple, la formule propositionnelle en forme normale disjonctive $(P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee (P_3 \wedge Q_3)$ a pour forme normale conjonctive

$$(P_1 \vee P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee Q_3) \wedge (P_1 \vee Q_2 \vee P_3) \wedge (P_1 \vee Q_2 \vee Q_3) \\ \wedge (Q_1 \vee P_2 \vee P_3) \wedge (Q_1 \vee P_2 \vee Q_3) \wedge (Q_1 \vee Q_2 \vee P_3) \wedge (Q_1 \vee Q_2 \vee Q_3).$$

Cela correspond à la forme clause

$$\{\{P_1, P_2, P_3\}, \{P_1, P_2, Q_3\}, \{P_1, Q_2, P_3\}, \{P_1, Q_2, Q_3\}, \\ \{Q_1, P_2, P_3\}, \{Q_1, P_2, Q_3\}, \{Q_1, Q_2, P_3\}, \{Q_1, Q_2, Q_3\}\}.$$

Sa généralisation $\bigvee_{1 \leq i \leq n} P_i \wedge Q_i$ a pour forme normale conjonctive $\bigwedge_{J \subseteq \{1, \dots, n\}} \bigvee_{1 \leq i \leq n} \ell_{J,i}$ où $\ell_{J,i} = P_i$ si $i \in J$ et $\ell_{J,i} = Q_i$ sinon; c'est une formule contenant 2^n clauses, chacune contenant *n* littéraux.

En conclusion, la mise sous forme normale conjonctive à l'aide de la distributivité de \vee sur \wedge produit une formule logiquement équivalente et est facile à utiliser sur de petits exemples « à la main », mais l'exemple 7.5 montre que cette transformation peut avoir un coût exponentiel, ce qui la rend inutilisable sur les formules propositionnelles que l'on souhaite manipuler en pratique. Ce problème a une solution : calculer une formule sous forme

normale conjonctive *équi-satisfiable* à la place d'une formule logiquement équivalente ; c'est ce que nous allons voir maintenant.

7.2.2. Forme clause équi-satisfiable. En général, étant donnée une formule propositionnelle φ en forme normale négative, on peut construire en temps linéaire une formule équi-satisfiable sous forme clause.

Pour chaque sous-formule φ' de φ , on introduit pour cela une proposition fraîche $Q_{\varphi'} \notin \text{Props}(\varphi)$ et on définit la formule propositionnelle

$$\psi_{\varphi'} \stackrel{\text{def}}{=} \begin{cases} P & \text{si } \varphi' = P, \\ \neg P & \text{si } \varphi' = \neg P, \\ Q_{\varphi_1} \vee Q_{\varphi_2} & \text{si } \varphi' = \varphi_1 \vee \varphi_2, \\ Q_{\varphi_1} \wedge Q_{\varphi_2} & \text{si } \varphi' = \varphi_1 \wedge \varphi_2. \end{cases}$$

La formule propositionnelle désirée est alors $\psi \stackrel{\text{def}}{=} Q_{\varphi} \wedge \bigwedge_{\varphi' \text{ sous-formule de } \varphi} (Q_{\varphi'} \rightarrow \psi_{\varphi'})$. Cette formule propositionnelle a deux propriétés remarquables :

- (1) elle est facile à mettre sous forme 3-CNF : c'est en effet une conjonction où les implications peuvent être mises sous forme normale conjonctive en utilisant la définition de l'implication et au besoin la distributivité de \vee sur \wedge : par exemple

$$(Q_{\varphi_1 \wedge \varphi_2} \rightarrow \psi_{\varphi_1 \wedge \varphi_2}) \leftrightarrow ((\neg Q_{\varphi_1 \wedge \varphi_2} \vee Q_{\varphi_1}) \wedge (\neg Q_{\varphi_1 \wedge \varphi_2} \vee Q_{\varphi_2}))$$

et

$$(Q_{\varphi_1 \vee \varphi_2} \rightarrow \psi_{\varphi_1 \vee \varphi_2}) \leftrightarrow (\neg Q_{\varphi_1 \vee \varphi_2} \vee Q_{\varphi_1} \vee Q_{\varphi_2}).$$

- (2) sa représentation arborescente est de *taille linéaire* en la taille de la formule φ : il y a une implication $Q_{\varphi'} \rightarrow \psi_{\varphi'}$ par sous-formule φ' de φ , et chacune de ces implications est de taille bornée par une constante.

Proposition 7.6. *Les formules propositionnelles φ et $\psi \stackrel{\text{def}}{=} Q_{\varphi} \wedge \bigwedge_{\varphi' \text{ sous-formule de } \varphi} (Q_{\varphi'} \rightarrow \psi_{\varphi'})$ sont équi-satisfiables.*

Démonstration. Supposons φ satisfaite par une interprétation I . On étend cette interprétation en associant, pour chaque sous-formule φ' , $\llbracket \varphi' \rrbracket^I$ à la proposition $Q_{\varphi'}$, ce qui définit une nouvelle interprétation $I' \stackrel{\text{def}}{=} I[\llbracket \varphi' \rrbracket^I / Q_{\varphi'}]_{\varphi' \text{ sous-formule de } \varphi}$. Montrons que $I' \models \psi$ et donc que ψ est satisfiable. Il suffit de montrer que chacune des clauses de ψ est satisfaite par I' .

Tout d'abord, comme $I \models \varphi$, $I' \models Q_{\varphi}$. Puis on montre par analyse de cas que, pour toute sous-formule φ' , on a $I' \models Q_{\varphi'} \rightarrow \psi_{\varphi'}$.

- cas $\varphi' = P$: on veut montrer que $I' \models Q_P \rightarrow P$. Supposons pour cela que $I' \models Q_P$. Alors par définition de I' , $I \models P$. Toujours par définition de I' , $I' \models P$ comme désiré.
- cas $\varphi' = \neg P$: on veut montrer que $I' \models Q_{\neg P} \rightarrow \neg P$. Supposons pour cela que $I' \models Q_{\neg P}$. Alors par définition de I' , $I \models \neg P$. Toujours par définition de I' , $I' \models \neg P$ comme désiré.
- cas $\varphi' = \varphi_1 \wedge \varphi_2$: on veut montrer que $I' \models Q_{\varphi'} \rightarrow Q_{\varphi_1} \wedge Q_{\varphi_2}$. Supposons pour cela que $I' \models Q_{\varphi'}$. Alors par définition de I' , $I \models \varphi'$, donc $I \models \varphi_1$ et $I \models \varphi_2$. Toujours par définition de I' , on a donc $I' \models Q_{\varphi_1}$ et $I' \models Q_{\varphi_2}$ comme désiré.
- cas $\varphi' = \varphi_1 \vee \varphi_2$: on fait une analyse similaire.

■ (PERIFEL, 2014, prop. 3-Z), (GOUBAULT-LARRECQ et MACKIE, 1997, exo. 2.23), (DAVID, NOUR et RAFFALLI, 2003, def. 7.4.15), (CARTON, 2008, p. 191), (HARRISSON, 2009, sec. 2.8)

☞ Dans le cadre de la mise sous forme clause, cette transformation est parfois appelée « transformation de TSEITIN », qui historiquement ne suppose pas φ sous forme normale négative et utilise des équivalences $Q_{\varphi'} \leftrightarrow \psi_{\varphi'}$ au lieu des implications $Q_{\varphi'} \rightarrow \psi_{\varphi'}$.

☞ La preuve de la proposition 7.6 montre en fait que si ψ est satisfiable, alors elle l'est par une extension d'une interprétation qui satisfait φ .

Inversement, supposons ψ satisfaite par une interprétation I' . On montre par induction sur les sous-formules φ' de φ que $I' \models Q_{\varphi'}$ implique $I' \models \varphi'$; comme $I' \models Q_{\varphi}$ on aura bien $I' \models \varphi$ et donc φ satisfiable.

Pour les cas de base $\varphi' = P$ (resp. $\varphi' = \neg P$), on a par hypothèse $I' \models Q_P \rightarrow P$ (resp. $I' \models Q_{\neg P} \rightarrow \neg P$) et donc $I' \models Q_P$ implique $I' \models P$ (resp. $I' \models \neg P$). Pour l'étape d'induction,

- si $\varphi' = \varphi_1 \wedge \varphi_2$, $I' \models Q_{\varphi'}$ implique $I' \models Q_{\varphi_1}$ et $I' \models Q_{\varphi_2}$ (car par hypothèse $I' \models Q_{\varphi'} \rightarrow (Q_{\varphi_1} \wedge Q_{\varphi_2})$), qui implique $I' \models \varphi_1$ et $I' \models \varphi_2$ (par hypothèse d'induction), qui implique $I' \models \varphi'$;
- si $\varphi' = \varphi_1 \vee \varphi_2$, on fait une analyse similaire. □

Corollaire 7.7. *Pour toute formule propositionnelle, on peut construire en temps déterministe linéaire une formule équi-satisfiable sous forme 3-CNF.*

Exemple 7.8. Reprenons la formule propositionnelle de l'exemple 7.5 pour $n = 3$: la formule $\bigvee_{1 \leq i \leq 3} P_i \wedge Q_i$ est représentée dans la figure 7.1, où l'on a annoté chacune des sous-formules avec des noms de propositions en orange.

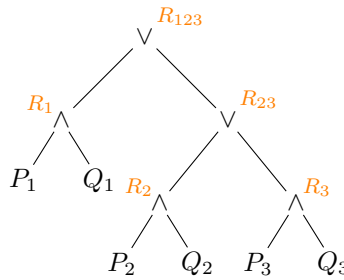


FIGURE 7.1. La formule propositionnelle $\bigvee_{1 \leq i \leq 3} P_i \wedge Q_i$ annotée.

Une formule propositionnelle en forme normale conjonctive équi-satisfiable (légèrement simplifiée par rapport à la transformation ci-dessus) est

$$R_{123} \wedge (\neg R_{123} \vee R_1 \vee R_{23}) \wedge (\neg R_{23} \vee R_2 \vee R_3) \wedge \bigwedge_{1 \leq i \leq 3} (\neg R_i \vee P_i) \wedge (\neg R_i \vee Q_i),$$

que l'on peut simplifier en

$$(R_1 \vee R_2 \vee R_3) \wedge \bigwedge_{1 \leq i \leq 3} (\neg R_i \vee P_i) \wedge (\neg R_i \vee Q_i).$$

La forme clausale correspondante est

$$\{\{R_1, R_2, R_3\}, \{\neg R_1, P_1\}, \{\neg R_1, Q_1\}, \{\neg R_2, P_2\}, \{\neg R_2, Q_2\}, \{\neg R_3, P_3\}, \{\neg R_3, Q_3\}\}.$$

■ (DUPARC, 2015, sec. I.2.10.1)

7.3. Forme normale disjonctive. Toujours en utilisant la distributivité, on obtient une mise sous *forme normale disjonctive* $\text{dnf}(\varphi)$ pour toute φ en forme normale négative; le cas le plus important de cette transformation est

$$\text{dnf}(\varphi \wedge (\psi \vee \psi')) \stackrel{\text{def}}{=} \text{dnf}((\psi \vee \psi') \wedge \varphi) \stackrel{\text{def}}{=} \text{dnf}(\varphi \wedge \psi) \vee \text{dnf}(\varphi \wedge \psi').$$

Le résultat est une formule propositionnelle $\bigvee_{1 \leq i \leq m} \bigwedge_{1 \leq j \leq n_i} \ell_{i,j}$ où les $\ell_{i,j}$ sont des littéraux.

Exemple 7.9. Considérons la formule propositionnelle

$$\neg(P \leftrightarrow (\neg Q \wedge R)) . \tag{7.1}$$

Sa forme normale négative est $(P \wedge (Q \vee \neg R)) \vee (\neg P \wedge \neg Q \wedge R)$. Une application de la distributivité de \wedge sur \vee fournit une formule logiquement équivalente sous forme normale disjonctive

$$(P \wedge Q) \vee (P \wedge \neg R) \vee (\neg P \wedge \neg Q \wedge R) . \tag{7.2}$$

Étant donnée une telle formule, déterminer si elle est satisfiable peut être effectué en temps linéaire (en supposant un hachage parfait des noms de propositions). En effet, une conjonction $M_i = \bigwedge_{1 \leq j \leq n_i} \ell_{i,j}$ est satisfiable si et seulement si elle ne contient pas à la fois une proposition P et sa négation $\neg P$; une formule $\bigvee_{1 \leq i \leq m} M_i$ est satisfiable si et seulement si au moins une des conjonctions M_i est satisfiable.

Comme la mise sous forme normale conjonctive, la mise sous forme normale disjonctive peut avoir un coût exponentiel. Cependant, et contrairement à ce que nous venons de voir pour la forme normale conjonctive en section 7.2.2, on ne peut pas espérer avoir un algorithme en temps polynomial pour calculer une formule propositionnelle sous forme normale disjonctive équi-satisfiable avec une formule donnée en entrée (sous réserve que $P \neq NP$), puisque résoudre la satisfiabilité de cette forme normale disjonctive se fait ensuite en temps polynomial.

☞ Les formes normales disjonctives pour des formules propositionnelles trouvent des applications en conception assistée par ordinateur, pour configurer des réseaux logiques programmables par exemple de type « PLA » ou « PAL », qui prennent la forme de disjonctions de conjonctions de littéraux, et peuvent être évoquées dans la leçon 19 « Fonctions et circuits booléens en architecture des ordinateurs ».

7.3.1. *Forme disjonctive complète.* Une alternative pour calculer une forme normale disjonctive d’une formule φ est de calculer sa table de vérité et d’en déduire sa *forme normale disjonctive complète* comme cela avait été fait dans la preuve du théorème 5.8 de complétude fonctionnelle. Formellement, φ est sous forme normale disjonctive complète si $\varphi = \bigvee_{I=\varphi} M_I$ où chaque conjonction $M_I = \bigwedge_{P \in \text{Props}(\varphi)} \ell_{I,P}$ liste pour chaque proposition $P \in \text{Props}(\varphi)$ les valeurs prises dans l’interprétation $I : \ell_{I,P} \stackrel{\text{def}}{=} P$ si $P^I = \mathbf{V}$ et $\ell_{I,P} \stackrel{\text{def}}{=} \neg P$ sinon.

Exemple 7.10. Reprenons la formule propositionnelle $\neg(P \leftrightarrow (\neg Q \wedge R))$ de l’exemple 7.9. Voici sa table de vérité.

P	Q	R	$\neg Q \wedge R$	$P \leftrightarrow (\neg Q \wedge R)$	$\neg(P \leftrightarrow (\neg Q \wedge R))$
V	V	V	F	F	V
V	V	F	F	F	V
V	F	V	V	V	F
V	F	F	F	F	V
F	V	V	F	V	F
F	V	F	F	V	F
F	F	V	V	F	V
F	F	F	F	V	F

Il y a quatre interprétations partielles de P , Q et R qui satisfont la formule : $[\mathbf{V}/P, \mathbf{V}/Q, \mathbf{V}/R]$, $[\mathbf{V}/P, \mathbf{V}/Q, \mathbf{F}/R]$, $[\mathbf{V}/P, \mathbf{F}/Q, \mathbf{F}/R]$ et $[\mathbf{F}/P, \mathbf{F}/Q, \mathbf{V}/R]$. La forme normale disjonctive complète de la formule est donc la disjonction suivante de quatre

avec son ensemble de littéraux, ce qui permet d'écrire par exemple $\ell \in M \subseteq M'$ pour ℓ un littéral et M, M' deux monômes. Le monôme vide est aussi noté \top .

Tout monôme M qui a pour conséquence logique une formule propositionnelle φ est appelé un *implicant* de φ . Si φ est sous forme normale disjonctive, alors chacun de ses monômes en est aussi un implicant, mais il peut y en avoir d'autres, comme par exemple $P \wedge \neg R$ pour la formule (7.3). Un implicant M est un implicant *premier* de φ si, pour tout monôme $M' \subsetneq M$, M' n'est pas un implicant de φ .

Dans le cas où la formule φ est déjà sous forme normale disjonctive, c'est-à-dire est une disjonction de monômes, alors on peut se concentrer sur les littéraux présents dans φ :

Propriété 7.11. *Soit D un ensemble de monômes. Si M est un implicant premier de $\bigvee_{M' \in D} M'$, alors $M \subseteq \bigcup_{M' \in D} M'$.*

Démonstration. Supposons par l'absurde qu'il existe un littéral $\ell \in M \setminus \bigcup_{M' \in D} M'$. On va montrer que le monôme $M \setminus \{\ell\}$ est aussi un implicant de $\bigvee_{M' \in D} M'$, ce qui contredira le fait que M est premier.

Soit donc I une interprétation telle que $I \models M \setminus \{\ell\}$. Alors $I[\mathbf{V}/\ell] \models M$. Comme M est un implicant de $\bigvee_{M' \in D} M'$, on a aussi $I[\mathbf{V}/\ell] \models \bigvee_{M' \in D} M'$. Il existe donc $M' \in D$ tel que $I[\mathbf{V}/\ell] \models M'$. Comme $\ell \notin M'$, on a aussi $I[\mathbf{F}/\ell] \models M'$ et donc $I \models M'$. Mais alors $I \models \bigvee_{M' \in D} M'$, contradiction. \square

La *forme normale première* d'une formule propositionnelle φ , aussi appelée « forme normale de BLAKE », est la disjonction de ses implicants premiers.

Exemple 7.12. Reprenons la formule $\neg(P \leftrightarrow (\neg Q \wedge R))$ de l'exemple 7.9 et sa forme normale disjonctive $(P \wedge Q) \vee (P \wedge \neg R) \vee (\neg P \wedge \neg Q \wedge R)$. Chacun de ses monômes en est aussi un implicant premier : elle est en fait sous forme normale première. Les deux monômes $P \wedge Q$ et $P \wedge \neg R$ en sont des implicants premiers car ni P , ni Q , ni $\neg R$ ne sont des implicants. Le monôme $\neg P \wedge \neg Q \wedge R$ est un implicant premier car ni $\neg P \wedge \neg Q$, ni $\neg P \wedge R$, ni $\neg Q \wedge R$ ne sont des implicants.

La forme normale première d'une formule propositionnelle est elle aussi une forme *canonique* pour la formule. Elle est nécessairement de taille inférieure ou égale à celle de la forme normale complète.

Couverture de monômes. Un ensemble D de monômes est une *couverture* d'un monôme M si M est un implicant de la disjonction $\bigvee_{M' \in D} M'$. C'est une *couverture minimale* si aucun sous-ensemble $D' \subsetneq D$ n'est une couverture de M .

Soit S un ensemble de littéraux sur \mathcal{P}_0 . L'ensemble des *littéraux purs* de S est l'ensemble des littéraux ℓ de S tels que leur négation $\bar{\ell}$ ne soit pas dans S :

$$\text{Pure}(S) \stackrel{\text{def}}{=} \{\ell \in S \mid \bar{\ell} \notin S\}. \quad (7.4)$$

Propriété 7.13. *Si D est une couverture minimale de M , alors*

$$\text{Pure}\left(\bigcup_{M' \in D} M'\right) = \bigcup_{M' \in D} M \cap M'.$$

Démonstration de l'inclusion \supseteq . Par l'absurde, supposons qu'il existe un littéral $\ell \in M$ qui ne soit pas pur dans D , c'est-à-dire que $\bar{\ell} \in M'' \in D$. Alors, pour toute interprétation I telle que $I \models M$, $I \models \bigvee_{M' \in D} M'$ puisque M est un implicant. Cependant, $I \models \ell$ et donc

■ Voir (KNUTH, 2008, exo. 32).

$I \not\models \bar{\ell}$ donc $I \not\models M''$. On a donc $I \models \bigvee_{M' \in D \setminus \{M''\}} M'$. Cela montre que $D \setminus \{M''\}$ est une couverture de M , ce qui contredit la minimalité de D . \square

Démonstration de l'inclusion \subseteq . Par l'absurde, supposons qu'il existe un littéral ℓ pur tel que $\ell \notin M$. Soient I une interprétation I telle que $I \models M$ et $D' \stackrel{\text{def}}{=} \{M' \in D \mid \ell \notin M'\} \subsetneq D$; on va montrer que $I \models \bigvee_{M' \in D'} M'$, ce qui contredira la minimalité de D .

Comme $\ell \notin M$, on a $I[\mathbf{F}/\ell] \models M$, et comme D couvre M , $I[\mathbf{F}/\ell] \models \bigcup_{M' \in D'} M'$. Comme ℓ est pur, on a aussi $\bar{\ell} \notin M'$ pour tout $M' \in D$; autrement dit, la proposition de ℓ n'apparaît pas dans D' . Alors par la propriété 5.3, on a aussi $I[\mathbf{V}/\ell] \models \bigvee_{M' \in D'} M'$. Or soit $I = I[\mathbf{V}/\ell]$, soit $I = I[\mathbf{F}/\ell]$, et dans tous les cas $I \models \bigvee_{M' \in D'} M'$, contradiction. \square

Corollaire 7.14 (nombre d'implicants premiers). *Soit D un ensemble de monômes. Alors il existe au plus $2^{|D|} - 1$ implicants premiers de $\bigvee_{M' \in D} M'$.*

Démonstration. Soit M un implicant premier de $\bigvee_{M' \in D} M'$. Alors il existe un sous-ensemble $D' \subseteq D$ non vide qui est une couverture minimale de M . Par la propriété 7.13, $M \cap (\bigcup_{M' \in D'} M') = \text{Pure}(\bigcup_{M' \in D'} M')$. Par la propriété 7.11, $M \subseteq \bigcup_{M' \in D'} M'$, et donc $M = \text{Pure}(\bigcup_{M' \in D'} M')$ est déterminé par ce sous-ensemble D' non vide.

Comme il y a $2^{|D|} - 1$ tels sous-ensembles D' , il y a au plus $2^{|D|} - 1$ implicants premiers de $\bigvee_{M' \in D} M'$. \square

■ C.f. (KNUTH, 2008, exo. 32).

Exemple 7.15. La borne supérieure du corollaire 7.14 est atteinte, par exemple par la formule sous forme normale disjonctive

$$\bigvee_{1 \leq i \leq n} P_i \wedge Q_1 \wedge \cdots \wedge Q_{i-1} \wedge \neg Q_i. \quad (7.5)$$

Pour tout ensemble non vide d'indices $J \subseteq \{1 \leq i \leq n\}$, on définit le monôme $M_J \stackrel{\text{def}}{=} \{P_i \mid i \in J\} \cup \{Q_1, \dots, Q_{(\max J)-1}, \neg Q_{\max J}\} \setminus \{Q_i \mid i \in J\}$. On peut noter qu'en particulier $M_{\{i\}} = P_i \wedge Q_1 \wedge \cdots \wedge Q_{i-1} \wedge \neg Q_i$ pour tout $1 \leq i \leq n$.

Si $I \models M_J$, soit j l'indice minimal de $\{1, \dots, n\}$ tel que $Q_j^I = \mathbf{F}$; comme $Q_{\max J}^I = \mathbf{F}$ on a $j \leq \max J$. Alors $I \models P_j \wedge Q_1 \wedge \cdots \wedge Q_{j-1} \wedge \neg Q_j$, donc M_J est bien un implicant.

Si $M \subsetneq M_J$,

– soit il existe un indice $j \in J$ tel que $P_j \notin M$ et alors il existe une interprétation I avec $P_j^I = Q_j^I = \mathbf{F}$ et $Q_i^I = \mathbf{V}$ pour tout $i < j$ qui satisfait M mais ne satisfait pas la formule (7.5) car, pour tout $1 \leq i \leq n$, $I \not\models P_i \wedge Q_1 \wedge \cdots \wedge Q_{i-1} \wedge \neg Q_i$:

pour $i < j$: puisque $Q_i^I = \mathbf{V}$,

pour $i = j$: puisque $P_j^I = \mathbf{F}$, et

pour $i > j$: puisque $Q_j^I = \mathbf{F}$;

– soit $\neg Q_{\max J} \notin M$ et alors l'interprétation I définie par $P_i^I = Q_i^I = \mathbf{V}$ pour tout $1 \leq i \leq n$ satisfait M mais ne satisfait pas la formule (7.5) car, pour tout $1 \leq i \leq n$, $I \not\models P_i \wedge Q_1 \wedge \cdots \wedge Q_{i-1} \wedge \neg Q_i$ puisque $Q_i^I = \mathbf{V}$;

– soit il existe un indice $j \notin J$, $j < \max J$, tel que $Q_j \notin M$ et alors l'interprétation I définie par $P_i^I = \mathbf{V}$ ssi $i \in J$ et $Q_i^I = \mathbf{V}$ ssi $i \notin \{j, \max J\}$ satisfait M mais ne satisfait la formule (7.5), car, pour tout $1 \leq i \leq n$, $I \not\models P_i \wedge Q_1 \wedge \cdots \wedge Q_{i-1} \wedge \neg Q_i$:

pour $i < j$: puisque $Q_i^I = \mathbf{V}$,

pour $i = j$: puisque $P_i^I = \mathbf{F}$ car $j \notin J$, et

pour $i > j$: puisque $Q_j^I = F$ et $j \leq i - 1$.

Cela montre que chaque M_J est premier.

Mise sous forme normale première. Le calcul des implicants premiers d'une formule sous forme normale disjonctive complète est l'objet de l'*algorithme de QUINE-McCLUSKEY*. On peut donner une version légèrement généralisée de cet algorithme qui prend en entrée une formule sous forme normale disjonctive (pas nécessairement complète). Un préalable à ces algorithmes est de savoir calculer les implicants premiers d'une disjonction de deux monômes.

■ Voir (KNUTH, 2008, exos. 29–31).

Lemme 7.16 (de consensus). *Soient M_1 et M_2 deux monômes et M un implicant premier de $M_1 \vee M_2$. Alors*

- (1) soit $M = M_1$ ou $M = M_2$,
- (2) soit $M = \text{Pure}(M_1 \cup M_2)$ et il existe un littéral ℓ tel que $M_1 \setminus M = \{\ell\}$ et $M_2 \setminus M = \{\bar{\ell}\}$.

Dans le cas (2), on appelle M le consensus de M_1 et M_2 et on le note $M_1 \sqcup M_2$.

Démonstration. Si l'ensemble $\{M_1, M_2\}$ n'est pas une couverture minimale de M , alors $M \models M_1$ ou $M \models M_2$. Mais alors $M_1 \subseteq M$ ou $M_2 \subseteq M$, et comme M était supposé premier, forcément $M_1 = M$ ou $M_2 = M$: nous sommes dans le cas (1).

Sinon, $\{M_1, M_2\}$ est une couverture minimale de M . Dans ce cas, par la propriété 7.13, $M \cap (M_1 \cup M_2) = \text{Pure}(M_1 \cup M_2)$. Comme M est supposé premier, par la propriété 7.11, $M \subseteq M_1 \cup M_2$, et donc $M = \text{Pure}(M_1 \cup M_2)$, ce qui correspond à la première condition du cas (2).

Si $M_1 \subseteq M$, on aurait $M \models M_1$, ce qui contredirait que $\{M_1, M_2\}$ est une couverture minimale; donc $M_1 \not\subseteq M$. Il existe donc un littéral $\ell \in M_1 \setminus M$. Comme $\ell \notin M = \text{Pure}(M_1 \cup M_2)$ et $\ell \in M_1$, on sait que $\bar{\ell} \in M_2 \setminus M$.

Nous avons presque vérifié la seconde condition du cas (2). Supposons par l'absurde qu'il existe un autre littéral dans cette situation, c'est-à-dire qu'il existe $\ell' \in (M_1 \setminus M) \setminus \{\ell, \bar{\ell}\}$. Encore une fois, $\bar{\ell}' \in M_2 \setminus M$. On montre alors que M ne peut pas être un implicant de $M_1 \vee M_2$. En effet, soit I une interprétation telle que $I \models M$. Comme aucun des littéraux $\ell, \bar{\ell}, \ell', \bar{\ell}'$ n'apparaît dans M , on a aussi $I[F/\ell, V/\ell'] \models M$. Cependant $I[F/\ell, V/\ell'] \not\models \ell$ donc $I[F/\ell, V/\ell'] \not\models M_1$, et $I[F/\ell, V/\ell'] \not\models \bar{\ell}'$ donc $I[F/\ell, V/\ell'] \not\models M_2$, ce qui montre que M n'est pas un implicant, contradiction. \square

Exemple 7.17 (consensus). Considérons l'ensemble de monômes $\{P \wedge Q \wedge R, P \wedge Q \wedge \neg R, P \wedge \neg Q \wedge \neg R, \neg P \wedge \neg Q \wedge R\}$ obtenu dans l'exemple 7.10. Alors $(P \wedge Q) = (P \wedge Q \wedge R) \sqcup (P \wedge Q \wedge \neg R)$ mais le monôme $P \wedge Q \wedge R$ n'est en consensus ni avec $P \wedge \neg Q \wedge \neg R$ ni avec $\neg P \wedge \neg Q \wedge R$. On a aussi $(P \wedge \neg R) = (P \wedge Q \wedge \neg R) \sqcup (P \wedge \neg Q \wedge \neg R)$ mais $P \wedge Q \wedge \neg R$ n'est pas en consensus avec $\neg P \wedge \neg Q \wedge R$, qui n'est d'ailleurs pas en consensus avec $P \wedge \neg Q \wedge \neg R$ non plus.

Nous sommes maintenant prêts à fournir un algorithme qui prend en entrée un ensemble D de monômes et retourne l'ensemble des implicants premiers de $\bigvee_{M' \in D} M'$.

☞ Voir KNUTH (2008, exo. 31) pour une version plus efficace de cet algorithme. L'algorithme de QUINE-McCLUSKEY en est une variante où D est supposé être une forme normale complète.

Fonction premiers(D)

```

1  $n := |\text{Props}(D)|$ ;  $j := n$ ;  $E_n := \min_{\subseteq} D$ 
2 tant que  $j > 0$  faire
3    $E_{j-1} := \emptyset$ ;
4   pour tous les  $M_1 \in E_j$  faire
5     pour tous les  $M_2 \in E_j$  si  $M_1 \sqcup M_2$  existe faire
6        $M := M_1 \sqcup M_2$ ;  $E_{j-1} := E_{j-1} \cup \{M\}$ 
7    $E_{j-1} := \min_{\subseteq}(E_j \cup E_{j-1})$ ;
8    $j := j - 1$ 
9 retourner  $E_0$ 

```

Proposition 7.18. *L'algorithme retourne l'ensemble des implicants premiers de l'ensemble de monômes D .*

Démonstration. La boucle principale est itérée $n = |\text{Props}(D)|$ fois; les boucles internes itèrent sur des paires $\{M_1, M_2\}$ de monômes issus d'ensembles E_j finis : l'algorithme termine bien.

Pour démontrer la correction de l'algorithme, on établit l'invariant suivant à la ligne 2 à l'entrée de la boucle principale.

Énoncé 7.18.1. Pour tout implicant M de $\bigvee_{M' \in D} M'$ avec $|M| \geq j$, il existe un implicant $M' \in E_j$ tel que $M' \subseteq M$.

On montre l'invariant par récurrence descendante sur $j \leq n$. Initialement, $j = n = |\text{Props}(D)|$, et donc tout implicant M de taille $|M| \geq n$ est un monôme complet avec n littéraux et doit contenir au moins un monôme de $E_n = \min_{\subseteq} D$. Puis, en cours d'exécution de l'algorithme pour $j < n$, soit M un implicant de $\bigvee_{M' \in D} M'$ avec $|M| \geq j$. Comme $j < n$, il existe une proposition $P \notin \text{Props}(M)$. Alors $M_1 \stackrel{\text{def}}{=} M \cup \{P\}$ et $M_2 \stackrel{\text{def}}{=} M \cup \{\neg P\}$ sont aussi des implicants de $\bigvee_{M' \in D} M'$, et sont tous deux de taille au moins $j + 1$. Par hypothèse de récurrence, il existe des implicants $M'_1, M'_2 \in E_{j+1}$ tels que $M'_1 \subseteq M_1$ et $M'_2 \subseteq M_2$ et on va raisonner sur le corps de la boucle pour $j + 1$. Si $M'_1 \subseteq M$ ou $M'_2 \subseteq M$ on aura alors cet implicant ou un de ses sous-ensembles ajouté dans E_j par la ligne 7. Sinon, $P \in M'_1$ et $\neg P \in M'_2$ et leur consensus $M'_1 \sqcup M'_2 \subseteq M$ est ajouté à E_j à la ligne 6 et lui ou un de ses sous-ensembles est bien conservé dans E_j ligne 7.

Par cet invariant, pour tout implicant premier M , il existe un implicant $M' \subseteq M$ tel que $M' \in E_0$ à la fin de l'exécution; comme M est supposé premier, nécessairement $M = M'$. Inversement, si $M \in E_0$ à l'issue de l'exécution, comme M est un implicant et comme il n'existe pas $M' \subsetneq M$ dans E_0 , c'est un implicant premier. \square

7.3.3. * *Forme normale disjonctive minimale.* On parle de forme normale disjonctive *minimale* pour une disjonction $\bigvee_{M' \in D} M'$ de monômes telle que $\sum_{M' \in D} |M'|$ soit minimale. Comme le montre la propriété suivante, une forme normale disjonctive minimale est nécessairement une disjonction d'implicants premiers.

Propriété 7.19. *La forme normale disjonctive minimale d'une formule propositionnelle φ est une disjonction d'implicants premiers de φ .*

▲ Le problème de minimisation d'une formule propositionnelle donnée sous forme normale disjonctive est Σ_2^P -complet (UMANS, 2001).

Démonstration. Écrivons tout d'abord la forme normale disjonctive minimale de φ comme la disjonction $\bigvee_{M' \in D} M'$ de son ensemble D de monômes. Chacun des monômes $M' \in D$ est un implicant de φ . Supposons par l'absurde que l'un d'entre eux ne soit pas premier et appelons-le M . Alors il existe un monôme $M'' \subsetneq M$ qui est un implicant de φ . On va montrer que φ est logiquement équivalente à $M'' \vee \bigvee_{M' \in D \setminus \{M\}} M'$, ce qui contredira la minimalité de D .

Montrons tout d'abord que $M'' \vee \bigvee_{M' \in D \setminus \{M\}} M'$ est une conséquence logique de la disjonction $\bigvee_{M' \in D} M'$. Comme $M'' \subsetneq M$, on a $M \models M''$. Supposons que $I \models \bigvee_{M' \in D} M'$ et faisons une distinction de cas : si $I \models M$ alors $I \models M''$ et donc $I \models M'' \vee \bigvee_{M' \in D \setminus \{M\}} M'$; sinon il existe $M' \in D \setminus \{M\}$ tel que $I \models M'$ et alors $I \models M'' \vee \bigvee_{M' \in D \setminus \{M\}} M'$.

Montrons maintenant que la disjonction $\bigvee_{M' \in D} M'$ est une conséquence logique de $M'' \vee \bigvee_{M' \in D \setminus \{M\}} M'$. Supposons que $I \models M'' \vee \bigvee_{M' \in D \setminus \{M\}} M'$ et faisons une distinction de cas : si $I \models M''$, comme c'est un implicant, $I \models \bigvee_{M' \in D} M'$; sinon il existe $M' \in D \setminus \{M\}$ tel que $I \models M'$ et alors $I \models \bigvee_{M' \in D} M'$. \square

Exemple 7.20. La forme normale première n'est pas forcément la forme normale disjonctive minimale. Considérons par exemple la formule propositionnelle sous forme normale disjonctive suivante

$$(P \wedge R) \vee (\neg P \wedge Q). \quad (7.6)$$

Chacun de ses deux monômes est un implicant premier. Cependant, ce ne sont pas les seuls : sa forme normale première est

$$(P \wedge R) \vee (Q \wedge R) \vee (\neg P \wedge Q). \quad (7.7)$$

Il existe un cas particulier où la forme normale première est minimale, qui est celui des formules propositionnelles sans négation.

Théorème 7.21 (QUINE). *Soit φ une formule propositionnelle sans négation. Alors la forme normale disjonctive minimale de φ est sa forme normale première.*

■ Voir (KNUTH, 2008, thm. Q).

Démonstration. Écrivons $\bigvee_{M' \in D} M'$ pour la forme normale disjonctive minimale de φ . Par la propriété 7.19, tous les monômes de $\bigvee_{M' \in D} M'$ sont des implicants premiers de φ . Nous allons montrer que tous les implicants premiers de φ doivent nécessairement apparaître dans D . Nous allons d'abord montrer le résultat suivant.

Énoncé 7.21.1. Soit φ une formule propositionnelle sans négation. Alors, pour tout implicant premier M de φ , M ne contient que des littéraux positifs.

Démonstration de l'énoncé 7.21.1. On commence par montrer que, si φ est une formule propositionnelle sans négation, alors $\llbracket \varphi \rrbracket$ est *monotone* au sens suivant : si $I \leq I'$ sont deux interprétations – où l'on considère l'ordre produit, c'est-à-dire $P^I \leq P^{I'}$ pour toute proposition $P \in \mathcal{P}_0$ avec \mathbb{B} ordonné par $\mathbf{F} < \mathbf{V}$ – alors $\llbracket \varphi \rrbracket^I \leq \llbracket \varphi \rrbracket^{I'}$. Cela se vérifie par induction sur les formules sans négation :

- pour le cas de base d'une proposition P , $\llbracket P \rrbracket^I = P^I \leq P^{I'} = \llbracket P \rrbracket^{I'}$ par hypothèse sur $I \leq I'$;
- pour l'étape d'induction sur une formule $\varphi \vee \psi$, $\llbracket \varphi \vee \psi \rrbracket^I = \llbracket \varphi \rrbracket^I$ ou $\llbracket \psi \rrbracket^I$ où par hypothèse d'induction $\llbracket \varphi \rrbracket^I \leq \llbracket \varphi \rrbracket^{I'}$ et $\llbracket \psi \rrbracket^I \leq \llbracket \psi \rrbracket^{I'}$ et donc, par monotonie de la fonction **ou**, $\llbracket \varphi \rrbracket^I$ ou $\llbracket \psi \rrbracket^I \leq \llbracket \varphi \rrbracket^{I'}$ ou $\llbracket \psi \rrbracket^{I'} = \llbracket \varphi \vee \psi \rrbracket^{I'}$;

- pour l'étape d'induction sur une formule $\varphi \wedge \psi$, on raisonne de même en utilisant la monotonie de la fonction **et**.

Revenons à l'énoncé 7.21.1. Supposons par l'absurde que M soit un implicant premier de φ et que $\neg P \in M$ pour une proposition $P \in \mathcal{P}_0$, alors on va montrer que $M \setminus \{\neg P\}$ est aussi un implicant de φ , ce qui contredira le fait que M était premier. En effet, si $I \models M \setminus \{\neg P\}$, alors $I[F/P] \models M$ et donc $I[F/P] \models \varphi$ puisque M est un implicant, mais aussi $I[V/P] \models \varphi$ par monotonie de $\llbracket \varphi \rrbracket$, et donc $I \models \varphi$. \square

Supposons maintenant par l'absurde qu'il existe un implicant premier M de φ qui ne soit pas dans D ; comme on vient de le voir avec l'énoncé 7.21.1, M est un ensemble de propositions. Considérons l'interprétation I définie par $P^I \stackrel{\text{def}}{=} \mathbf{V}$ pour tout $P \in M$ et $Q^I \stackrel{\text{def}}{=} \mathbf{F}$ pour tout $Q \notin M$. Alors $I \models M$ et donc $I \models \varphi$ puisque M est un implicant. Mais $I \not\models \bigvee_{M' \in D} M'$, car sinon il y aurait $M' \in D$ tel que $I \models M'$ et comme M' est un implicant premier et ne contient donc pas de littéral négatif, nécessairement $M' \subseteq M$, ce qui est impossible : on a supposé $M \notin D$ donc $M' \subsetneq M$ mais cela contredit le fait que M soit un implicant premier. On vient de montrer que $\bigvee_{M' \in D} M'$ n'était pas logiquement équivalente à φ , ce qui est absurde. \square

Exemple 7.22. Voyons un exemple d'application du théorème 7.21 de QUINE. Considérons la formule propositionnelle

$$\bigwedge_{1 \leq i \leq n} P_i \vee Q_i. \quad (7.8)$$

Pour tout sous-ensemble d'indices $J \subseteq \{1, \dots, n\}$, le monôme $M_J \stackrel{\text{def}}{=} \{P_i \mid i \in J\} \cup \{Q_i \mid 1 \leq i \leq n, i \notin J\}$ est un implicant de la formule. En effet, si $I \models M_J$, alors $P_i^I = \mathbf{V}$ pour tout $i \in J$ et $Q_i^I = \mathbf{V}$ pour tout $1 \leq i \leq n$ avec $i \notin J$. Donc pour tout $1 \leq i \leq n$, I satisfait la clause $P_i \vee Q_i$: I satisfait donc la formule.

De plus, si M est un implicant premier de la formule, alors il existe un ensemble d'indices J tel que $M_J \subseteq M$. En effet, d'après le propriété 7.11 et l'énoncé 7.21.1, M est un sous-ensemble de $\{P_i, Q_i \mid 1 \leq i \leq n\}$. Si on suppose par l'absurde que $M_J \not\subseteq M$ pour tout ensemble d'indices J , alors il existe $1 \leq i \leq n$ tel que $P_i \notin M$ et $Q_i \notin M$. Mais alors on peut définir une interprétation I par $P^I \stackrel{\text{def}}{=} \mathbf{V}$ pour tout $P \in M$ et $Q^I \stackrel{\text{def}}{=} \mathbf{F}$ pour tout $Q \notin M$. Cette interprétation est telle que $I \models M$ mais $I \not\models P_i \vee Q_i$, et donc I ne satisfait pas la formule (7.8) : cela contredit le fait que M est un implicant.

Dès lors, les monômes M_J sont exactement les implicants premiers de la formule (7.8), et par le théorème 7.21, la formule $\bigvee_{J \subseteq \{1, \dots, n\}} M_J$ en est la forme normale disjonctive minimale, qui est de taille $n \cdot 2^n$.

■ (CHINAL, 1967, sec. 10.4.1)

▲ À ne pas confondre avec l'algorithme de QUINE de recherche de modèle par simplification de la section 8.3.2.

Algorithme de QUINE-MCCLUSKEY. On a vu jusqu'ici que l'on pouvait calculer la forme normale disjonctive première par l'algorithme premiers, et qu'une forme normale disjonctive minimale était constituée d'un sous-ensemble de ses monômes (propriété 7.19). L'algorithme de QUINE-MCCLUSKEY cherche à déterminer une telle forme minimale :

- (1) calculer la forme normale disjonctive complète D à l'aide de la table de vérité de la formule,
- (2) calculer la forme normale disjonctive première D' en appelant premiers(D),

- (3) calculer pour chaque implicant premier $M \in D'$ une couverture minimale $D_M \subseteq D$ de M (qui n'est autre que $D_M = \{M' \in D \mid M \subseteq M'\}$ puisque D est sous forme disjonctive complète),
- (4) résoudre (de manière heuristique) l'instance du problème de COUVERTURE D'ENSEMBLE défini par D et la famille de sous-ensembles $\{D_M\}_{M \in D'}$, c'est-à-dire chercher à trouver un sous-ensemble $D'' \subseteq D'$ de taille $\sum_{M \in D''} |M|$ minimale tel que $D = \bigcup_{M \in D''} D_M$.

☞ *Le problème COUVERTURE D'ENSEMBLE est NP-complet (KARP, 1972; GAREY et JOHNSON, 1979, [SP5]). Il s'agit ici d'une version pondérée du problème, où chaque ensemble D_M a un poids $|M|$.*

Exemple 7.23. Appliquons l'algorithme de QUINE-McCLUSKEY à la formule de l'exemple 7.20.

- (1) L'ensemble de ses monômes complets est

$$D = \{P \wedge Q \wedge R, P \wedge \neg Q \wedge R, \neg P \wedge Q \wedge R, \neg P \wedge Q \wedge \neg R\}.$$

- (2) L'ensemble de ses monômes premiers est

$$D' = \{P \wedge R, Q \wedge R, \neg P \wedge Q\}.$$

- (3) On a

$$\begin{aligned} D_{P \wedge R} &= \{P \wedge Q \wedge R, P \wedge \neg Q \wedge R\}, \\ D_{Q \wedge R} &= \{P \wedge Q \wedge R, \neg P \wedge Q \wedge R\}, \\ D_{\neg P \wedge Q} &= \{\neg P \wedge Q \wedge R, \neg P \wedge Q \wedge \neg R\}. \end{aligned}$$

comme ensembles de monômes complets de D qui couvrent minimalement chaque monôme de D' .

- (4) On a $D = D_{P \wedge R} \cup D_{\neg P \wedge Q}$ et l'ensemble de monômes

$$D'' = \{P \wedge R, \neg P \wedge Q\}$$

correspond à une forme normale disjonctive minimale $(P \wedge R) \vee (\neg P \wedge Q)$.

7.4. * Diagrammes binaires de décision. Pour finir cette section sur les formes normales en logique propositionnelle, nous allons voir une représentation à l'aide de diagrammes binaires de décision, qui fournissent des représentations *canoniques* pour les formules booléennes.

▀ (GOUBAULT-LARRECQ et MACKIE, 1997, sec. 1.4.4), (LASSAIGNE et ROUGEMONT, 2004, sec. 1.3.5), (HARRISSON, 2009, sec. 2.11), (JAUME et al., 2020, sec. 7.2.2)

On définit pour cela une *formule de SHANNON* comme une formule sur la syntaxe abstraite

$$\beta ::= P? \beta : \beta \mid \top \mid \perp \quad (\text{formule de SHANNON})$$

où P est une proposition de \mathcal{P}_0 . Une formule de SHANNON est une formule propositionnelle de forme particulière, où $P? \beta_{\top} : \beta_{\perp}$ (lu « si P alors β_{\top} sinon β_{\perp} ») est du sucre syntaxique pour $(P \rightarrow \beta_{\top}) \wedge (\neg P \rightarrow \beta_{\perp})$.

Comme pour les formules propositionnelles, une formule de SHANNON peut être représentée (voir figure 1.1)

- sous forme arborescente, auquel cas on parlera d'*arbre binaire de décision*, abrégé par *BDT* pour « *binary decision tree* », ou
- sous forme de circuit partageant les sous-formules isomorphes, auquel cas on parlera de *diagramme binaire de décision*, abrégé par *BDD* pour « *binary decision diagram* ».

☞ *On ne parlera pas ici de l'algorithmique des BDD, sujet très riche susceptible d'être abordé dans la leçon 4 « Exemples de structures de données. Applications. » ; voir (KNUTH, 2011, sec. 7.1.4).*

Dans ces représentations, plutôt que d'utiliser des sommets de degré sortant 3 vers P , β_{\top} et β_{\perp} , on étiquette plutôt le sommet par P avec deux arcs sortants, l'un plein vers β_{\top} et l'autre pointillé vers β_{\perp} ; voir figure 7.3.

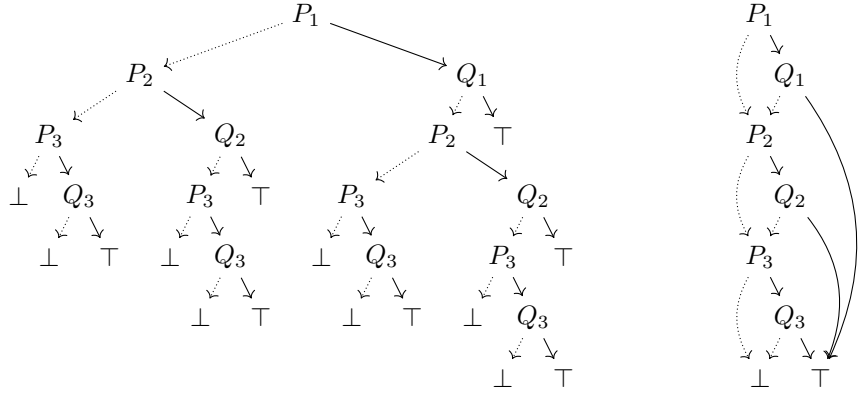


FIGURE 7.3. Un BDT et un BDD pour $(P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee (P_3 \wedge Q_3)$.

Toute formule propositionnelle φ peut être exprimée à l'aide d'un BDD équivalent : si $P \in \mathcal{P}_0$, alors φ est équivalente à $P?\varphi[\top/P] : \varphi[\perp/P]$ par le lemme 6.7 de substitution propositionnelle; il suffit d'itérer ce processus sur les propositions de $\text{Props}(\varphi)$. Le coût de cette transformation peut être exponentiel.

7.4.1. *Formules de SHANNON ordonnées.* Fixons un ordre total $<$ sur l'ensemble de propositions \mathcal{P}_0 . Une formule de SHANNON est *ordonnée* par $<$ si, pour toute sous-formule $P?\beta_{\top} : \beta_{\perp}$ et pour toute proposition $Q \in \text{Props}(\beta_{\top}) \cup \text{Props}(\beta_{\perp})$, $P < Q$. En d'autres termes, l'ordre dans lequel les propositions sont testées est le même pour toutes les branches de la formule. Un BDD ordonné est aussi appelé un *OBDD*. On peut noter que le BDT et le BDD de la figure 7.3 sont en fait ordonnés.

En particulier, dans une formule de SHANNON ordonnée, chaque proposition ne peut être testée qu'au plus une fois le long d'une branche : une telle branche décrit donc une interprétation partielle. Cela signifie que la formule est satisfiable si et seulement si elle a une branche qui finit par \top , ce qui peut se tester en temps linéaire par un algorithme de parcours de graphe.

7.4.2. *Formules de SHANNON complètes.* Un cas particulier des formule de SHANNON ordonnées est celui des formules *complètes* sur un ensemble de propositions $P_1 < \dots < P_n$: toutes ces propositions apparaissent dans l'ordre dans toutes les branches. Un OBDD complet est aussi appelé un *QDD*. La figure 7.4 montre le QDD correspondant à la formule de la figure 7.3.

Proposition 7.24 (borne supérieure). *Soit β un QDD sur n propositions. Alors $|\beta| \leq \frac{2^n}{n}$.*

Démonstration. Pour tout $k \leq n$, on construit un graphe acyclique dirigé pour une formule de SHANNON complète β_k équivalente à β ; en identifiant les sous-graphes isomorphes de

☞ Cette complexité linéaire de la satisfiabilité indique que la mise sous forme d'OBDD doit être non polynomiale, sans quoi on pourrait résoudre SAT dans P. Il est en fait inutile de supposer $P \neq NP$ pour cela : il existe une famille de fonctions ISA_n sur $n + \lg n$ variables pour lesquelles il existe des BDT (non ordonnés) de taille $n^2 / \lg n$ (et donc des formules propositionnelles de taille $O(n^2 / \lg n)$), mais qui nécessitent des OBDD de taille au moins $2^{(n / \lg n) - 3}$ (BREITBART, HUNT III et ROSENKRANTZ, 1995, thm. 3).

■ (WEGENER, 2000, thm. 2.3.1).

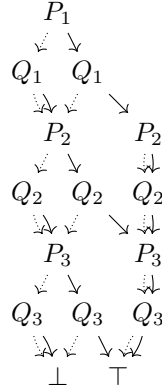


FIGURE 7.4. Le QDD pour $(P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee (P_3 \wedge Q_3)$.

☞ Un QDD β sur $P_1 < \dots < P_n$ est essentiellement isomorphe à l'automate déterministe complet minimal pour $L(\beta) \stackrel{\text{def}}{=} \{I \in \mathbb{B}^n \mid I \models \beta\}$ – aux transitions sortantes des sommets \perp et \top près (WEGENER, 2000, sec. 3.2). Cela montre que la représentation par QDD est elle aussi canonique.

β_k , on obtient alors le QDD β . On choisit alors une valeur de k montrant que la taille de β_k , et donc de β , est bornée par $\frac{2^n}{n}$.

Le graphe de β_k est constitué de deux « étages ».

- (1) L'un pour les variables P_i pour $i \leq k$: on développe pour cela le BDT complet de profondeur k et donc de taille $2^k - 1$.
- (2) Le second étage est constitué de BDT complets de profondeur $n - k$, enracinés dans le premier étage, mais où l'on a identifié les BDT de hauteur $n - k$ isomorphes. Il y a $2^{2^{n-k}}$ fonctions booléennes sur $n - k$ variables, chacune ayant un BDT complet de taille $2^{n-k} - 1$.

☞ Il existe des fonctions booléennes à n arguments dont le plus petit BDD (et donc le plus petit OBDD et donc le plus petit QDD) est de taille au moins $\frac{2^n}{n}(1 - \varepsilon_n)$ où $\lim_{n \rightarrow \infty} \varepsilon_n = 0$ (BREITBART, HUNT III et ROSENKRANTZ, 1995, thm. 1), montrant que la borne supérieure de la proposition 7.24 est optimale.

Le graphe de β_k est donc de taille bornée par $2^k + 2^{n-k} \cdot 2^{2^{n-k}}$. Posons maintenant $k \stackrel{\text{def}}{=} n - \lfloor \lg(n - \lg n) \rfloor$. On a alors une borne sur la taille de β_k :

$$\begin{aligned}
 |\beta_k| &= 2^{n - \lfloor \lg(n - \lg n) \rfloor} + 2^{\lfloor \lg(n - \lg n) \rfloor} \cdot 2^{2^{\lfloor \lg(n - \lg n) \rfloor}} \\
 &\leq 2^{n - \lfloor \lg(n - \lg n) \rfloor} + 2^{\lfloor \lg(n - \lg n) \rfloor} \cdot 2^{n - \lg n} && \text{(car } 2^x \text{ est croissante)} \\
 &= 2^n \cdot \left(2^{-\lfloor \lg(n - \lg n) \rfloor} + 2^{\lfloor \lg(n - \lg n) \rfloor - \lg n} \right) \\
 &\leq 2^n \cdot \left(2^{-\lfloor \lg(n - \lg n) \rfloor} \cdot 2^{\lfloor \lg(n - \lg n) \rfloor - \lg n} \right) && \text{(car } 2^x \text{ est convexe)} \\
 &= 2^n \cdot 2^{-\lg n} . && \square
 \end{aligned}$$

7.4.3. *Formules de SHANNON réduites.* Une formule de SHANNON est *réduite* si pour toute sous-formule $P? \beta_{\top} : \beta_{\perp}$, les formules β_{\top} et β_{\perp} ne sont pas logiquement équivalentes. Un OBDD réduit est aussi appelé un ROBDD. On peut observer que le BDT et le BDD de la figure 7.3 sont en fait réduits. À noter qu'un ROBDD est valide si et seulement s'il est égal à \top , et insatisfiable si et seulement s'il est égal à \perp .

Le principal résultat dans le cadre de la leçon 28 est l'unicité des ROBDD.

Théorème 7.25 (canonicité). *Soient β et β' deux ROBDD construits pour le même ordre $<$ sur \mathcal{P}_0 . Alors β et β' sont logiquement équivalents si et seulement s'ils sont égaux.*

■ (GOUBAULT-LARRECQ et MACKIE, 1997, thm. 2.48), (LASSAIGNE et ROUEMONT, 2004, cor. 1.4), (WEGENER, 2000, thm. 3.1.4 et 3.3.2)

Démonstration. Si β et β' sont égaux, alors ils sont clairement équivalents. Pour la réciproque, on montre par récurrence sur $|\text{Props}(\beta)| + |\text{Props}(\beta')|$ que si β et β' sont logiquement équivalents, alors $\beta = \beta'$.

Pour le cas de base au rang $n = 0$, β et β' doivent être la même constante \perp ou \top .

Pour l'étape de récurrence, soit P la proposition minimale de $\text{Props}(\beta) \cup \text{Props}(\beta')$. Sans perte de généralité, on va supposer que $P \in \text{Props}(\beta)$. Comme $P = \min_{<} \text{Props}(\beta)$, $\beta = P? \beta_{\top} : \beta_{\perp}$.

Si $P \notin \text{Libres}(\beta')$: alors β' , β_{\top} et β_{\perp} sont logiquement équivalents, ce qui contredit le fait que β était réduit. En effet, si $I \models \beta'$, alors par la propriété 5.3, $I[\mathbf{V}/P] \models \beta'$. Par équivalence de β et β' , $I[\mathbf{V}/P] \models \beta$, et donc $I[\mathbf{V}/P] \models \beta_{\top}$ et comme $P \notin \text{Props}(\beta_{\top})$ par la propriété 5.3 $I \models \beta_{\top}$. Inversement, si $I \models \beta_{\top}$, alors $I[\mathbf{V}/P] \models \beta_{\top}$ par la propriété 5.3, donc $I[\mathbf{V}/P] \models \beta$; comme $\beta \equiv \beta'$, $I[\mathbf{V}/P] \models \beta'$ et par la propriété 5.3 $I \models \beta'$. Le même raisonnement vaut pour β_{\perp} .

Si $P \in \text{Libres}(\beta')$: alors $\beta' = P? \beta'_{\top} : \beta'_{\perp}$ et on observe que β_{\top} et β'_{\top} sont équivalents et β_{\perp} et β'_{\perp} aussi; par hypothèse de récurrence on a alors $\beta_{\top} = \beta'_{\top}$ et $\beta_{\perp} = \beta'_{\perp}$ et donc $\beta = \beta'$.

Détaillons l'observation pour le cas de β'_{\top} . Si $I \models \beta_{\top}$, comme $P \notin \text{Props}(\beta_{\top})$, par la propriété 5.3, $I[\mathbf{V}/P] \models \beta_{\top}$ et donc $I[\mathbf{V}/P] \models \beta$. Par suite, $I[\mathbf{V}/P] \models \beta'$, donc $I[\mathbf{V}/P] \models \beta'_{\top}$ et $P \notin \text{Props}(\beta'_{\top})$, $I \models \beta'_{\top}$. La direction opposée, tout comme le cas de $\beta_{\perp} \equiv \beta'_{\perp}$, est similaire. \square

Étant donné un OBDD non réduit, on peut le *réduire* de haut en bas : si on rencontre $P? \beta : \beta$ (le test d'égalité est en temps constant grâce au partage des sous-formules), on remplace ce sous-BDD par β . On peut observer que ce processus appliqué à l'OBDD de la figure 7.4 construit le ROBDD de la figure 7.3. Ce processus montre aussi que pour un ordre donné, le ROBDD d'une formule est son OBDD de taille minimale.

🗨️ *Changer l'ordre des variables peut avoir des conséquences dramatiques sur la taille des OBDD, voir (WEGENER, 2000, ch. 5) et (KNUTH, 2011, sec. 7.1.4).*

7.4.4. Lien avec les automates finis. Rappelons que \sqsubseteq dénote l'ordre d'extension sur les interprétations partielles. Une interprétation partielle I est *adéquate* pour une formule φ si, pour toute extension $I' \sqsupseteq I$, $I' \models \varphi$. Clairement, si I est une interprétation totale telle que $I \models \varphi$, alors sa restriction de domaine $\text{Props}(\varphi)$ est adéquate par la propriété 5.3. Cependant, cette restriction n'est pas nécessairement une interprétation adéquate *minimale* pour \sqsubseteq : par exemple, $[\mathbf{V}/P, \mathbf{V}/Q]$ est adéquate pour $(P \vee \neg P) \wedge Q$, mais $[\mathbf{V}/Q]$ l'est aussi. On définit donc l'ensemble des interprétations adéquates minimales de φ comme $\text{MinSat}(\varphi) \stackrel{\text{def}}{=} \min_{\sqsubseteq} \{I \mid \forall I' \sqsupseteq I, I' \models \varphi\}$.

Fixons un ordre total sur $\text{Props}(\varphi) = \{P_1 < \dots < P_n\}$. On considère l'alphabet $\Sigma \stackrel{\text{def}}{=} \{P_i, \neg P_i \mid 1 \leq i \leq n\}$ des littéraux sur $\text{Props}(\varphi)$. À une interprétation partielle I avec $\text{dom}(I) = \{P_{i_1} < \dots < P_{i_k}\} \subseteq \text{Props}(\varphi)$, on associe le mot $w_I \stackrel{\text{def}}{=} a_{i_1} \dots a_{i_k}$ sur Σ , défini par $a_{i_j} \stackrel{\text{def}}{=} P_{i_j}$ si $I \models P_{i_j}$ et $a_{i_j} \stackrel{\text{def}}{=} \neg P_{i_j}$ sinon. On définit alors le langage $L(\varphi) \stackrel{\text{def}}{=} \{w_I \mid I \in \text{MinSat}(\varphi)\}$.

Proposition 7.26. *Soit φ une formule propositionnelle. Alors $\text{BDD}(\varphi)$ est isomorphe à l'automate minimal de $L(\varphi)$.*

8. LE PROBLÈME DE SATISFIABILITÉ

Résumé. De nombreux problèmes informatiques, et en particulier tous les problèmes NP-complets, peuvent être exprimés comme la satisfiabilité d'une formule propositionnelle de taille polynomiale. Les *solveurs SAT* sont des logiciels dédiés à ce problème, qui prennent en entrée une formule propositionnelle sous forme normale conjonctive (écrite au *format DIMACS*), et cherchent à répondre si la formule est satisfiable ou non.

Une manière d'implémenter un solveur SAT est d'utiliser l'*algorithme DPLL* dû à DAVIS, PUTNAM, LOGEMANN et LOVELAND. Cet algorithme travaille par *simplification* d'ensembles de clauses : la simplification d'un ensemble de clauses par un littéral ℓ élimine les clauses qui contiennent ℓ et retire $\bar{\ell}$ des clauses restantes. L'algorithme DPLL simplifie en priorité par les littéraux *unitaires* (ℓ est unitaire s'il existe une clause qui ne contient que ce littéral) et les littéraux *purs* (ℓ est pur si le littéral $\bar{\ell}$ n'apparaît nulle part dans l'ensemble de clauses).

Tout problème de décision dans NP a une réduction polynomiale vers SAT, ce qui permet de traduire l'entrée du problème en une formule propositionnelle et de tenter de le résoudre à l'aide d'un solveur SAT. Deux exemples : la résolution de dépendances logicielles (section 8.4) et la coloration de graphe (section 8.5).

8.1. **Le problème SAT.** Le problème de *satisfiabilité* est le problème de décision suivant.

Problème (SAT).

instance : une formule propositionnelle φ

question : φ est-elle satisfiable ?

Ce problème est clairement dans NP : il suffit en effet de « deviner » non déterministiquement une interprétation $I \in \mathbb{B}^{\text{Props}(\varphi)}$ telle que $I \models \varphi$. Une telle interprétation est de taille polynomiale, et vérifier $I \models \varphi$ se fait en temps polynomial. La difficulté est de montrer que SAT est NP-difficile.

Théorème 8.1 (COOK-LEVIN). *SAT est NP-complet.*

Dans de nombreuses utilisations de SAT, on suppose de plus que la formule φ est sous forme k -clausale pour un certain k fixé. Le problème de décision associé est k SAT.

Problème (k SAT).

instance : un ensemble fini F de k -clauses propositionnelles

question : F est-il satisfiable ?

■ Voir (KNUTH, 2008, sec. 7.1.1) et (KNUTH, 2015, sec. 7.2.2.2) pour une présentation approfondie des aspects algorithmiques du problème de satisfiabilité.

☞ Plusieurs problèmes de décision de cette section peuvent aussi servir dans la leçon 26 « Classes P et NP. Problèmes NP-complets. Exemples ».

■ (PERIFEL, 2014, thm. 3-V), (ARORA et BARAK, 2009, thm. 2.10), (PAPADIMITRIOU, 1993, thm. 8.2), (CARTON, 2008, thm. 4.19), (LASSAIGNE et ROUGEMONT, 2004, thm. 11.1), (GAREY et JOHNSON, 1979, thm. 2.1); la preuve du théorème de COOK-LEVIN sera vue dans le cadre de la préparation en calculabilité et complexité.

Corollaire 8.2. k SAT est NP-complet pour tout $k \geq 3$.

Démonstration. C'est une conséquence immédiate du théorème de COOK-LEVIN combiné au corollaire 7.7. \square

☞ Puisque tout problème NP-complet a une réduction en temps polynomial vers SAT, on dispose donc avec ces solveurs d'implémentations souvent très efficaces en pratique pour quantité de problèmes difficiles (voir la longue liste du GAREY et JOHNSON) : on parle de « révolution SAT »!

8.2. Solveurs SAT. Les solveurs SAT sont des programmes qui déterminent si une formule propositionnelle φ donnée est satisfiable ou non, et si oui, fournissent une interprétation I qui satisfait la formule, c'est-à-dire telle que $I \models \varphi$.

8.2.1. \square Format DIMACS. Les solveurs SAT emploient un format de fichier standard pour les formules propositionnelles en forme clausale, appelé « format DIMACS » du nom du *Center for Discrete Mathematics and Theoretical Computer Science* qui avait organisé les premières compétitions internationales. Voici par exemple comment représenter la forme clausale de l'exemple 7.8 en format DIMACS :

```
example-7-8.cnf
c exemple 7.8 en format DIMACS
c table des propositions
c R1 P1 Q1 R2 P2 Q2 R3 P3 Q3
c 1 2 3 4 5 6 7 8 9
c
p cnf 9 7
1 4 7 0
-1 2 0
-1 3 0
-4 5 0
-4 6 0
-7 8 0
-7 9 0
```

Le fichier commence par cinq lignes de commentaires, qui débutent par le caractère « c ». En format DIMACS, les propositions sont représentées par des entiers strictement positifs, et on a ajouté en commentaire comment les noms des propositions de l'exemple 7.8 ont été numérotés pour faciliter la lecture de l'exemple (mais rien n'impose de le faire). La sixième ligne est le véritable début du fichier, qui commence par « p cnf » suivi de deux nombres :

- « 9 » correspond au nombre de propositions utilisées dans la forme clausale ;
- « 7 » correspond au nombre de clauses.

Les clauses occupent les lignes suivantes. Chaque clause se termine par le caractère « 0 », et consiste en une séquence de nombres entiers non nuls : un nombre positif n désigne la proposition numérotée par n , tandis qu'un nombre négatif $-n$ désigne la négation de cette proposition. Par exemple, à la huitième ligne, « -1 » correspond à $\neg R_1$, tandis que « 2 » correspond à P_1 .

8.2.2. \square Utilisation d'un solveur SAT. MINISAT (<http://minisat.se/>) est un solveur SAT facile à installer puisqu'il existe des paquets pour distributions GNU/Linux. Un exemple d'invocation sur le fichier DIMACS de la section 8.2.1 est d'appeler la commande « minisat exemple-7-8.cnf exemple-7-8.modele ». Dans le cas où l'ensemble de clauses fourni en entrée est satisfiable, une interprétation partielle est retournée :

exemple-7-8.modele

SAT

-1 2 3 -4 5 6 7 8 9 0

L'interprétation est retournée sous la forme d'une clause DIMACS, où les propositions associées à **V** apparaissent comme des entiers positifs et celles associées à **F** comme des entiers négatifs. En l'occurrence, l'interprétation en termes de l'exemple 7.8 est

$$[F/R_1, V/P_1, V/Q_1, F/R_2, V/P_2, V/Q_2, V/R_3, V/P_3, V/Q_3].$$

En terme de la formule originale de l'exemple 7.5, nous avons vu dans la preuve de la proposition 7.6 qu'il suffit d'ignorer les propositions fraîches ajoutées par la mise sous forme équi-satisfiable; l'interprétation partielle suivante est donc un modèle de $(P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee (P_3 \wedge Q_3)$:

$$[V/P_1, V/Q_1, V/P_2, V/Q_2, V/P_3, V/Q_3].$$

8.3. Recherche de modèle. Cette section présente une des approches possibles pour résoudre le problème de satisfiabilité, qui a été utilisée jusque dans les années 1990 dans les solveurs SAT, et qui est conceptuellement assez simple.

8.3.1. Recherche de modèle par énumération. On peut résoudre automatiquement le problème SAT : puisque par la propriété 5.3, il suffit de trouver une interprétation partielle de domaine $\text{Props}(\varphi)$ qui satisfait φ , on peut simplement énumérer les $2^{|\text{Props}(\varphi)|}$ interprétations possibles. En termes de tables de vérité, cela revient à construire la table de φ et de tester si au moins une ligne met φ à **V**.

Exemple 8.3. Reprenons la forme clausale de l'exemple 5.12

$$F = \{\{P, Q, \neg R\}, \{Q, R\}, \{\neg P, \neg Q, R\}, \{\neg P, \neg R\}, \{P, \neg Q\}\}.$$

Sa table de vérité est donnée dans la table 8.1.

TABLE 8.1. La table de vérité de l'ensemble de clauses de l'exemple 5.12.

P	Q	R	$\{P, Q, \neg R\}$	$\{Q, R\}$	$\{\neg P, \neg Q, R\}$	$\{\neg P, \neg R\}$	$\{P, \neg Q\}$	F
V	V	V	V	V	V	F	V	F
V	V	F	V	V	F	V	V	F
V	F	V	V	V	V	F	V	F
V	F	F	V	F	V	V	V	F
F	V	V	V	V	V	V	F	F
F	V	F	V	V	V	V	F	F
F	F	V	F	V	V	V	V	F
F	F	F	V	F	V	V	V	F

On voit dans cette table que chaque ligne, c'est-à-dire chaque interprétation partielle de domaine $\{P, Q, R\}$, contient au moins une entrée **F** pour une des clauses de F . Par conséquent, la colonne pour F ne contient que des **F** : cette forme clausale est insatisfiable.

☞ On ne connaît pas d'algorithme pour SAT qui travaille en temps polynomial dans le pire des cas – on soupçonne même qu'un tel algorithme en temps sous-exponentiel n'existe pas, ce qui est appelé l'« exponential time hypothesis ».

Plutôt que d'écrire explicitement la table de vérité, on peut aussi représenter les interprétations partielles de domaine $\text{Props}(\varphi)$ sous la forme d'un arbre : pour chaque proposition $P \in \text{Props}(\varphi)$, on branche sur les deux choix $I \models \neg P$ et $I \models P$. L'arbre correspondant pour l'exemple 8.3 est donné dans la figure 8.1.

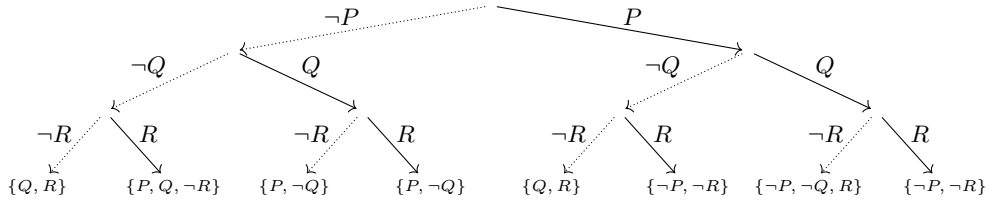


FIGURE 8.1. Un arbre de recherche de modèle pour l'exemple 8.3.

Chaque branche de l'arbre décrit une interprétation partielle sous la forme d'une liste de littéraux, et l'on a décoré chaque feuille de l'arbre par une clause non satisfaite par cette interprétation ; par exemple, la branche la plus à gauche étiquetée par $\neg P, \neg Q, \neg R$ correspond à l'interprétation partielle $[F/P, F/Q, F/R]$ (qui est la dernière ligne de la table 8.1), et cette interprétation ne satisfait pas la clause $\{Q, R\}$.

On peut observer que cet arbre de recherche est un arbre sémantique au sens de la section 6.5.2.

Cette représentation sous forme d'arbre de recherche a deux intérêts :

- d'une part, elle est moins longue à écrire à la main qu'une table de vérité,
- d'autre part, elle suggère un algorithme récursif qui explore l'arbre pour tester si une formule sous forme clause est satisfiable ou non : chaque nœud interne de l'arbre correspond à un appel récursif d'une fonction pour tester la satisfiabilité, et aux feuilles on vérifie s'il existe au moins une clause satisfaite par l'interprétation partielle pour cette branche.

À noter que, dans le cas de formules propositionnelles sous forme clause, il est très aisé de vérifier si une clause C est satisfaite ou non par une interprétation : $I \models C$ si et seulement s'il existe $\ell \in C$ tel que $I \models \ell$. Cela suggère le pseudo-code suivant pour évaluer la valeur de vérité d'une clause C sous une interprétation I :

```

Fonction evalueclause(C, I)
  1 pour tous les  $\ell \in C$  faire
  2    $\lfloor$  si  $I \models \ell$  alors retourner  $\vee$ 
  3 retourner  $\text{F}$ 

```

Par suite, pour une formule propositionnelle sous forme clause, c'est-à-dire un ensemble de clauses F , $I \models F$ si et seulement si $\forall C \in F, I \models C$:

```

Fonction evalueclausal(F, I)
  1 pour tous les  $C \in F$  faire
  2    $\lfloor$  si non evalueclause(C, I) alors retourner  $\text{F}$ 
  3 retourner  $\vee$ 

```

Le pseudo-code qui suit décrit l'algorithme récursif suggéré par les arbres de recherche, qui prend en entrée un ensemble de clauses F et une interprétation partielle I de domaine initialement vide.

```

Fonction satisfiable( $F, I$ )
1 si dom( $I$ ) = Props( $F$ ) alors
2   retourner evaluateclausal( $F, I$ )
3 sinon
4   choisir  $P \in \text{Props}(F) \setminus \text{dom}(I)$ 
5   retourner satisfiable( $F, I[V/P]$ ) ou satisfiable( $F, I[\bar{V}/P]$ )

```

Cette implémentation naïve d'un solveur SAT suffit pour des petits exemples, mais prend plus d'une seconde sur ma machine pour résoudre le problème de coloriage de la section 8.5.

8.3.2. *Recherche de modèle par simplification : l'algorithme de QUINE.* Un défaut de l'algorithme par énumération de la section précédente est qu'il attend d'avoir construit une interprétation partielle de domaine Props(φ) avant de tester si les clauses sont satisfaites. Pourtant, il est parfois possible de répondre plus tôt : par exemple, dans l'arbre de la figure 8.1, le nœud atteint en suivant le chemin $\neg P, Q$ correspond à une interprétation partielle $[F/P, V/Q]$ qui ne satisfait pas la clause $\{P, \neg Q\}$, quel que soit le choix de l'interprétation de la proposition R . On aurait pu interrompre la recherche de modèle plus tôt.

Cependant, évaluer toutes les clauses C d'une formule à chaque nœud interne de l'arbre d'interprétation partielle I telle que Props(C) \subseteq dom(I) serait coûteux. À la place, l'idée de l'algorithme de QUINE est une recherche de modèle par *simplification* de l'ensemble des clauses au fur et à mesure de l'exploration de l'arbre.

Simplification de formes clauseales. La recherche de modèle par simplification simplifie un ensemble de clauses propositionnelles S jusqu'à obtenir une clause vide – auquel cas S n'était pas satisfiable – ou un ensemble vide de clauses – auquel cas S était satisfiable. Soit S un ensemble de clauses. On définit la *simplification* de S par un littéral ℓ comme l'ensemble de clauses

$$S[\top/\ell] \stackrel{\text{def}}{=} \{C \mid ((C \cup \{\bar{\ell}\}) \in S \text{ ou } \bar{\ell} \notin C \in S) \text{ et } \ell \notin C\}$$

où l'on a éliminé les clauses de S contenant ℓ et simplifié les clauses de S de la forme $C \cup \{\bar{\ell}\}$ en leur enlevant $\bar{\ell}$. Par exemple,

$$\begin{aligned} \{\{P, Q\}, \{\neg P, R\}, \{P, \neg P\}\}[\top/P] &= \{\{R\}\}, \\ \{\{P, Q\}, \{\neg P, R\}, \{P, \neg P\}\}[\top/\neg P] &= \{\{Q\}\}. \end{aligned}$$

La simplification revient effectivement à substituer \top à ℓ et \perp à $\bar{\ell}$ dans toutes les clauses de S et à simplifier le résultat en utilisant les équivalences logiques $\varphi \vee \top \leftrightarrow \top$, $\varphi \vee \perp \leftrightarrow \varphi$ et $\varphi \wedge \top \leftrightarrow \varphi$. La simplification d'un ensemble F de clauses par un littéral ℓ s'écrit en pseudo-code comme suit.

Fonction $\text{simplifie}(F, \ell)$

- 1 $F' := \emptyset$
- 2 **pour tous les** $C \in F$ **faire**
- 3 \lfloor **si** $\ell \notin C$ **alors** $F' := F' \cup \{C \setminus \{\bar{\ell}\}\}$
- 4 **retourner** F'

Pour une interprétation I , on note $I[V/\ell]$ pour l'interprétation qui associe V à P si $\ell = P$, F à P si $\ell = \neg P$, et Q^I à toute proposition $Q \notin \text{Props}(\ell)$.

Recherche par simplification. L'algorithme de QUINE fait une *recherche par simplification* : on cherche à trouver une interprétation qui satisfait toutes les clauses d'un ensemble F de clauses, en testant successivement si $F[\top/P]$ ou $F[\top/\neg P]$ est satisfiable ; voir la figure 8.2 (où les littéraux colorés en orange sont impactés par la prochaine simplification).

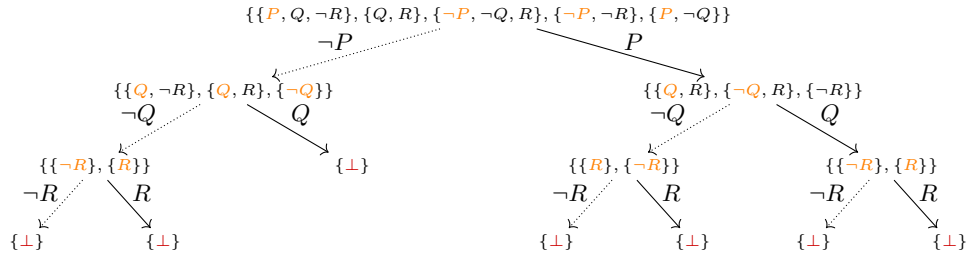


FIGURE 8.2. Un arbre de recherche par simplification pour l'exemple 8.3.

Cette recherche *réussit* s'il existe une branche qui termine sur l'ensemble vide \emptyset de clauses, et *échoue* si toutes les branches aboutissent à un ensemble qui contient la clause vide (celle-ci est notée \perp). La recherche illustrée dans la figure 8.2 échoue : l'ensemble F de l'exemple 8.3 était bien insatisfiable.

Exemple 8.4. Voici un autre exemple de recherche par simplification. Soit l'ensemble de clauses $F \stackrel{\text{def}}{=} \{\{P, \neg R, \neg P\}, \{Q, \neg R\}, \{-Q\}\}$. Un exemple de recherche par simplification est donné dans la figure 8.3.

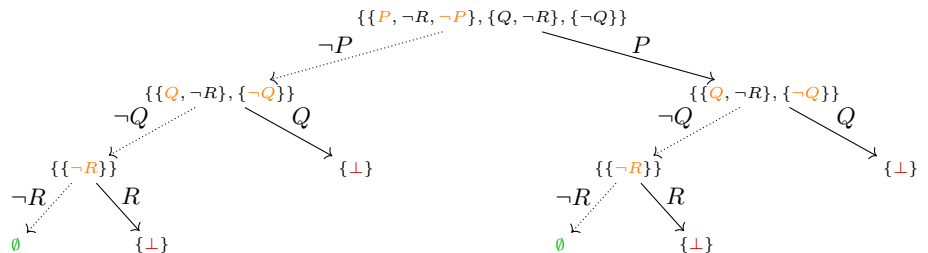


FIGURE 8.3. Un arbre de recherche par simplification pour l'exemple 8.4.

Cette recherche réussit, puisqu'il existe au moins une feuille étiquetée par l'ensemble vide de clauses, par exemple la feuille atteinte par le chemin $P, \neg Q, \neg R$; et en effet, toute interprétation qui étend $[V/P, F/Q, F/R]$ est un modèle de F .

Voici un pseudo-code pour une implémentation récursive de l'algorithme de QUINE de recherche par simplification.

Fonction satisfiable(F)

```

1 si  $F = \emptyset$  alors retourner  $\mathbb{V}$ 
2 si  $\perp \in F$  alors retourner  $\mathbb{F}$ 
3 choisir  $P \in \text{Props}(F)$ 
4 retourner
   satisfiable(simplifie( $F, P$ )) ou satisfiable(simplifie( $F, \neg P$ ))

```

Remarque 8.5. À noter dans ce pseudo-code que le choix de la proposition à utiliser pour simplifier à la ligne 3 n'a pas besoin d'être le même le long de toutes les branches. Par exemple, la figure 8.4 montre une recherche par simplification pour l'exemple 8.3 qui échoue plus rapidement que celle de la figure 8.2. On obtient ici un arbre de recherche par simplification isomorphe à l'arbre sémantique fermé de la figure 6.2.

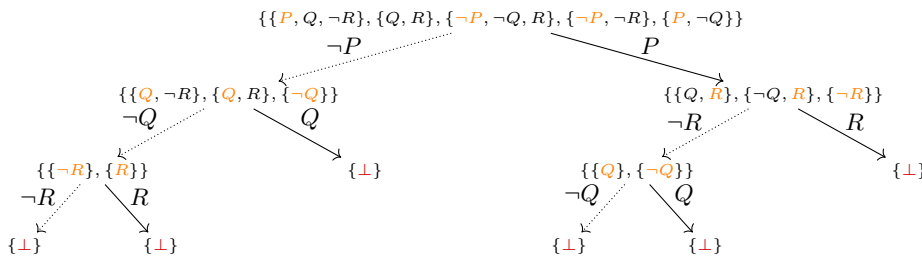


FIGURE 8.4. Un autre arbre de recherche par simplification pour l'exemple 8.3.

Exemple 8.6. Considérons la formule sous forme clausale

$$\{\{P, Q, R\}, \{\neg P, Q, R\}, \{R\}, \{P, \neg Q, \neg R\}, \{\neg P, \neg Q, \neg R\}, \{\neg R\}\}.$$

Si on effectue les simplifications sur P puis Q puis R , l'arbre de recherche obtenu est celui de la figure 8.5, qui est aussi grand que celui d'une recherche par énumération exhaustive.

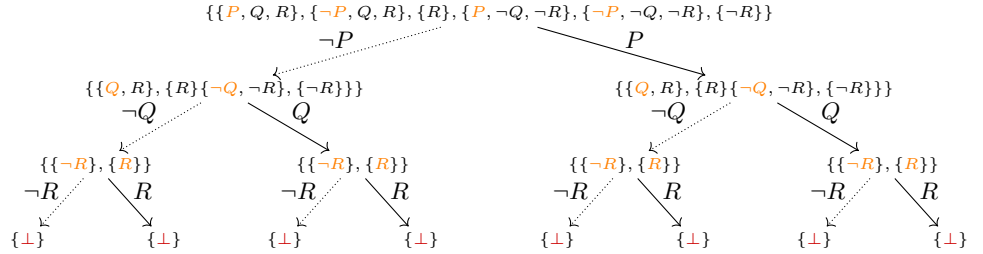


FIGURE 8.5. Un arbre de recherche par simplification pour l'exemple 8.6.

Il est pourtant possible de faire une recherche par simplification beaucoup plus efficace, en commençant par la proposition R , comme illustré dans la figure 8.6. En général, il peut être avantageux de simplifier en priorité sur des propositions qui apparaissent dans beaucoup de clauses.

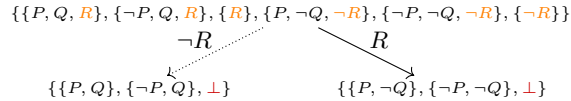


FIGURE 8.6. Un autre arbre de recherche par simplification pour l'exemple 8.6.

Correction et complétude. L'algorithme de QUINE de recherche de modèle par simplification peut se comprendre par le biais de règles de réécriture qui agissent sur des ensembles de clauses. Les règles (split_P) et ($\text{split}_{\neg P}$) de la figure 8.7 cherchent à montrer qu'un ensemble fini F est satisfiable en le réduisant à l'ensemble vide de clauses. Les arbres des figures 8.2 à 8.6 recensent des réécritures possibles par ce système de règles.

$$\begin{array}{lll}
 F \rightarrow_{\text{spl}} F[\top/P] & \text{où } P \in \text{Props}(F) & (\text{split}_P) \\
 F \rightarrow_{\text{spl}} F[\top/\neg P] & \text{où } P \in \text{Props}(F) & (\text{split}_{\neg P})
 \end{array}$$

FIGURE 8.7. Règles de réécriture de la recherche par simplification.

Il est aisé de voir que F est satisfiable si et seulement si $F \rightarrow_{\text{spl}}^* \emptyset$ à l'aide des règles (split_P) et ($\text{split}_{\neg P}$) pour $P \in \text{Props}(F)$. En effet, notons $I[\mathbf{V}/\ell]$ pour l'interprétation qui associe \mathbf{V} à P si $\ell = P$, \mathbf{F} à P si $\ell = \neg P$, et Q^I à toute proposition $Q \notin \text{Props}(\ell)$. On a alors la conséquence suivante du lemme 6.7 de substitution propositionnelle

Propriété 8.7. Pour toute ensemble S de clauses, toute interprétation I et tout littéral ℓ , $I \models S[\top/\ell]$ si et seulement si $I[\mathbf{V}/\ell] \models S$.

Une autre remarque importante est que la simplification par un littéral P ou $\neg P$ conduit à un ensemble de clauses où ni P ni $\neg P$ n'apparaît.

Propriété 8.8. Soit S un ensemble de clauses et P une proposition. Alors $P \notin \text{Props}(S[\top/P])$ et $P \notin \text{Props}(S[\top/\neg P])$.

Comme les règles de réécriture de la figure 8.7 n'appliquent la simplification qu'à une proposition $P \in \text{Props}(F)$, on a que $F \rightarrow_{\text{spl}} F'$ implique $\text{Props}(F) \supseteq \text{Props}(F')$, donc on ne peut appliquer les règles qu'au plus $|\text{Props}(F)|$ fois à un ensemble F donné : on dit que ce système de règles de réécriture *termine*.

On en déduit le théorème suivant des propriétés 8.7 et 8.8.

Théorème 8.9. Soit F un ensemble fini de clauses. Alors les règles de la figure 8.7 sont correctes et complètes : F est satisfiable si et seulement si $F \rightarrow_{\text{spl}}^* \emptyset$.

Démonstration. Pour la correction, c'est-à-dire pour montrer que $F \rightarrow_{\text{spl}}^* \emptyset$ implique F satisfiable, il suffit d'observer d'une part que l'ensemble vide de clauses \emptyset est satisfiable (il est même valide), et d'autre part que si F' est satisfiable – disons par une interprétation I telle que $I \models F'$ – et $F \rightarrow_{\text{spl}} F'$ par une des règles de la figure 8.7, alors F est satisfiable :

- pour (split_P) : alors $F' = F[\top/P]$ pour la proposition P : on a $I[\mathbf{V}/P] \models F$ par la propriété 8.7 ;
- pour $(\text{split}_{\neg P})$: alors $F' = F[\top/\neg P]$ pour la proposition P : on a $I[\mathbf{F}/P] \models F$ par la propriété 8.7.

Une simple récurrence sur le nombre de réécritures dans $F \rightarrow_{\text{spl}}^* \emptyset$ démontre alors la correction.

Pour la complétude, c'est-à-dire pour montrer que F satisfiable implique $F \rightarrow_{\text{spl}}^* \emptyset$, supposons que $I \models F$. On ordonne $\text{Props}(F)$ de manière arbitraire comme $P_1 < \dots < P_n$.

Soit $F_0 \stackrel{\text{def}}{=} F$; on applique pour chaque $1 \leq i \leq n$ à la proposition P_i sur l'ensemble F_{i-1}

- soit (split_{P_i}) si $I \models P_i$ et alors $F_i \stackrel{\text{def}}{=} F_{i-1}[\top/P_i]$ et comme $I = I[\mathbf{V}/P_i]$, $I \models F_i$ par la propriété 8.7 ;
- soit $(\text{split}_{\neg P_i})$ si $I \models \neg P_i$ et alors $F_i \stackrel{\text{def}}{=} F_{i-1}[\top/\neg P_i]$ et comme $I = I[\mathbf{V}/\neg P_i]$, $I \models F_i$ par la propriété 8.7.

Comme $\text{Props}(F_i) = \{P_{i+1}, \dots, P_n\}$ pour tout $0 \leq i \leq n$ par la propriété 8.8, F_n est un ensemble de clauses sans propositions. De plus, il ne peut pas contenir la clause vide puisque $I \models F_n$. Donc $F_n = \emptyset$. \square

8.3.3. * *Algorithme de DAVIS, PUTNAM, LOGEMANN et LOVELAND.* Couramment appelé *DPLL* d'après ses inventeurs, cet algorithme sert d'inspiration aux solveurs SAT actuels. L'algorithme *DPLL* est un raffinement de l'algorithme de *QUINE* de recherche de modèle par simplification vu dans la section 8.3.2.

L'idée de départ de l'algorithme *DPLL* provient de la remarque 8.5 : comment choisir les littéraux par lesquels simplifier l'ensemble de clauses ? Comme vu dans l'exemple 8.6, un mauvais choix va résulter en un arbre de recherche très grand, potentiellement aussi grand que l'arbre de recherche par énumération naïve de la section 8.3.1 ; à l'inverse, un bon choix peut mener beaucoup plus rapidement à la réponse. L'algorithme *DPLL* fournit des heuristiques pratiques pour choisir ces littéraux. Voici deux telles heuristiques, qui identifient les littéraux *unitaires* et les littéraux *purs* de l'ensemble de clauses.

Littéraux unitaires. Dans un ensemble S de clauses, une clause C est *unitaire* si elle ne contient qu'un seul littéral, et on dit alors que ce littéral est *unitaire*. Formellement, ℓ est

■ (CONCHON et SIMON, 2018, sec. 2.2.3), (GOUBAULT-LARRECQ et MACKIE, 1997, sec. 2.4.3), (HARRISSON, 2009, sec. 2.9)

unitaire dans S si $S = S' \cup \{\{\ell\}\}$ pour un ensemble S' . Dans tout modèle I de S , c'est-à-dire si $I \models S$, un littéral unitaire ℓ sera nécessairement satisfait par $I : I \models \ell$. C'est donc un bon choix pour une simplification.

Par exemple, dans l'ensemble de clauses de l'exemple 8.6, les littéraux R et $\neg R$ sont unitaires.

Littéraux purs. On définit de manière similaire à l'équation (7.4) l'ensemble des littéraux *purs* d'un ensemble S de clauses comme l'ensemble des littéraux ℓ tels que $\bar{\ell}$ n'apparaît dans aucune clause de S :

$$\text{Pur}(S) \stackrel{\text{def}}{=} \{\ell \mid \forall C \in S. \bar{\ell} \notin C\}.$$

Dans tout modèle I de S , c'est-à-dire si $I \models S$, si ℓ est un littéral pur de S , alors $I[\mathbf{V}/\ell]$ est encore un modèle de S . C'est encore une fois un bon choix pour une simplification.

Exemple 8.10. Soit l'ensemble de clauses

$$F \stackrel{\text{def}}{=} \{\{P, \neg Q, R\}, \{P, \neg R\}, \{Q, R\}, \{P, \neg Q\}\}.$$

Alors $\text{Pur}(F) = \{P\}$.

Correction et complétude. L'algorithme DPLL peut être vu comme une extension du système de règles de la figure 8.7, qui introduit deux nouvelles règles (unit) et (pure). Tout comme lors de la recherche de modèle par simplification, les règles de la figure 8.8 cherchent à montrer qu'un ensemble fini F de clauses est satisfiable en le réduisant à l'ensemble vide.

$$\begin{array}{llll} F \cup \{\{\ell\}\} \rightarrow_{\text{dpll}} F[\top/\ell] & & & \text{(unit)} \\ F \rightarrow_{\text{dpll}} F[\top/\ell] & \text{où } \ell \in \text{Pur}(F) & & \text{(pure)} \\ F \rightarrow_{\text{dpll}} F[\top/P] & \text{où } P \in \text{Props}(F) & & \text{(split}_P\text{)} \\ F \rightarrow_{\text{dpll}} F[\top/\neg P] & \text{où } P \in \text{Props}(F) & & \text{(split}_{\neg P}\text{)} \end{array}$$

FIGURE 8.8. Règles de réécriture de DPLL.

On montre que ce système de règles reste correct ; la complétude découle directement de celle du système de recherche de modèle par simplification.

Théorème 8.11. *Soit F un ensemble fini de clauses. Alors les règles de la figure 8.8 sont correctes et complètes : F est satisfiable si et seulement si $F \rightarrow_{\text{dpll}}^* \emptyset$.*

Démonstration. Pour la correction, puisque la recherche de modèle par simplification était correcte (voir le théorème 8.9), il suffit de montrer que, si F' est satisfiable – disons par une interprétation I telle que $I \models F'$ – et $F \rightarrow_{\text{dpll}} F'$ par (unit) et (pure), alors F est satisfiable :

- pour (unit) : alors $F' = F''[\top/\ell]$ et $F = F'' \cup \{\{\ell\}\}$ pour un littéral unitaire ℓ de F'' : on a $I[\mathbf{V}/\ell] \models F''$ par la propriété 8.7 et donc $I[\mathbf{V}/\ell] \models F$;
- pour (pure) : alors $F' = F[\top/\ell]$ pour un littéral pur ℓ : on a $I[\mathbf{V}/\ell] \models F$ par la propriété 8.7.

Pour la complétude, c'est-à-dire pour montrer que F satisfiable implique $F \rightarrow_{\text{dpll}}^* \emptyset$, cela découle du théorème 8.9. En effet, si F est satisfiable, alors $F \rightarrow_{\text{spl}}^* \emptyset$, c'est-à-dire $F \rightarrow_{\text{dpll}}^* \emptyset$ en n'utilisant que les règles (split_P) et (split_{¬P}). \square

Algorithme DPLL. L'intérêt des règles (unit) et (pure) est qu'elles sont *inversibles*, au sens suivant.

Proposition 8.12 (inversibilité). *Soient F et F' deux ensembles de clauses. Si F est satisfiable et $F \xrightarrow{\text{(unit)}}_{\text{dpll}} F'$ ou $F \xrightarrow{\text{(pure)}}_{\text{dpll}} F'$, alors F' est aussi satisfiable.*

Démonstration. Supposons que I soit une interprétation telle que $I \models F$.

- Pour (unit) : alors $F = F \cup \{\{\ell\}\}$ et en particulier $I \models \ell$. Donc $I[\mathbb{V}/\ell] = I \models F$ et par la propriété 8.7, $I \models F[\top/\ell] = F'$.
- Pour (pure) : alors $F' = F[\top/\ell]$ où ℓ est un littéral dans $\text{Pur}(F)$. Si $I \models \ell$, alors $I[\mathbb{V}/\ell] = I \models F$ et par la propriété 8.7 $I \models F[\top/\ell] = F'$. Inversement, si $I \not\models \ell$, alors comme $I \models F$, dans toutes les clauses $C \in F$, il existe un littéral $\ell_C \in C$ tel que $I \models \ell_C$ et forcément $\ell_C \neq \ell$ pour toutes les clauses $C \in F$. Mais alors $I[\mathbb{V}/\ell] \models \ell_C$ pour toutes les clauses $C \in F$, et donc $I[\mathbb{V}/\ell] \models F$. Par la propriété 8.7, $I \models F[\top/\ell] = F'$. \square

L'inversibilité et la correction des règles (unit) et (pure) signifie que, si $F \xrightarrow{\text{(unit)}}_{\text{dpll}} F'$ ou $F \xrightarrow{\text{(pure)}}_{\text{dpll}} F'$, alors F est satisfiable si et seulement si F' l'est. Autrement dit, il n'est pas utile d'introduire des *points de choix* comme lors de l'utilisation des règles (split_P) et (split_{¬P}), pour revenir en arrière s'il s'avérait que F' était insatisfiable. Les règles (unit) et (pure) peuvent donc être appliquées arbitrairement. La stratégie usuelle de l'algorithme DPLL est de les appliquer en priorité (et dans cet ordre) avant d'essayer (split_P) ou (split_{¬P}), de manière à accélérer la recherche de preuve en éliminant des points de choix. Voici le pseudo-code de l'algorithme écrit en style récursif.



Fonction satisfiable(F)

- 1 **si** $F = \emptyset$ **alors retourner** \mathbb{V}
- 2 **si** $\perp \in F$ **alors retourner** \mathbb{F}
- 3 **si** $\exists \ell. F = F \cup \{\{\ell\}\}$ **alors retourner** satisfiable(simplifie(F, ℓ))
- 4 **si** $\exists \ell \in \text{Pur}(F)$ **alors retourner** satisfiable(simplifie(F, ℓ))
- 5 **choisir** $P \in \text{Props}(F)$
- 6 **retourner**
satisfiable(simplifie(F, P)) **ou** satisfiable(simplifie($F, \neg P$))

Comme dans le cas de la recherche par simplification, les réécritures du système de la figure 8.8 sont de longueur au plus $|\text{Props}(F)|$.

Reprenons les différents exemples de cette section. Pour l'exemple de la figure 8.4, l'algorithme DPLL construit (entre autres) l'arbre de recherche de la figure 8.9. Pour l'exemple 8.4, l'arbre de recherche DPLL est celui de la figure 8.10. Pour l'exemple des figures 8.5 et 8.6, l'algorithme DPLL construit (entre autres) l'arbre de recherche de la figure 8.11. Pour l'exemple 8.10, l'algorithme DPLL construit l'arbre de recherche de la figure 8.12.

8.4. Application : résolution de dépendances logicielles. Voici un problème concret en informatique : comment assurer que toutes les dépendances d'un logiciel soient correctement installées et configurées ? Cela touche les programmes installés sur un système, par exemple via un gestionnaire de paquets comme apt sur les systèmes Debian Linux et apparentés (comme Ubuntu), mais aussi les bibliothèques nécessaires pour compiler du code,

 L'algorithme DPLL est implémenté sous forme itérative en utilisant du backtracking. Les performances des solveurs SAT actuels tiennent à une gestion fine des retours aux points de choix, et lors de ces retours à l'ajout de nouvelles clauses inférées à partir de la branche d'échec, ceci afin de guider la recherche vers une branche de succès; cette technique est appelée « conflict-driven clause learning ».  Voir (CONCHON et SIMON, 2018, sec. 2.2) et (ZHANG et al., 2001).

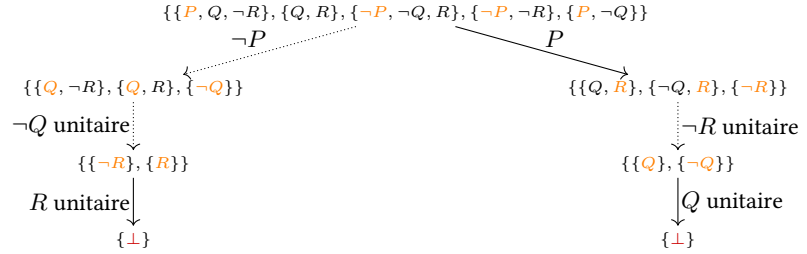


FIGURE 8.9. Un arbre de recherche DPLL pour l'exemple 8.3.

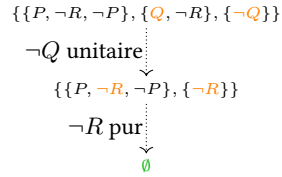


FIGURE 8.10. L'arbre de recherche DPLL pour l'exemple 8.4.

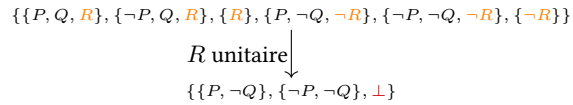


FIGURE 8.11. Un arbre de recherche DPLL pour l'exemple 8.6.

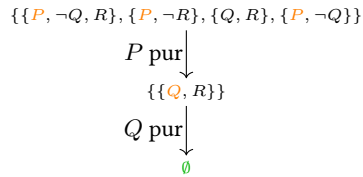



FIGURE 8.12. L'arbre de recherche DPLL pour l'exemple 8.10.

par exemple via `autoconf` pour C/C++, `maven` pour Java, `opam` pour OCaml, `pip` pour Python, etc. Des problèmes surgissent assez vite avec tous ces systèmes du fait de *conflicts* entre certaines versions de dépendances qui ne peuvent pas être installées simultanément ; on parle couramment de « *dependency hell* ».

Comme nous allons le voir, on peut résoudre ces problèmes de dépendances à l'aide d'un modélisation en logique propositionnelle et d'un solveur SAT ; c'est d'ailleurs ce qui est fait au sein du logiciel Eclipse à l'aide du solveur `Sat4j` pour la gestion des plugins.

☞ *On ne va pas donner la formulation générale du problème de résolution de dépendances ici, mais il est NP-complet (MANCINELLI et al., 2006).*

8.4.1.  *Modélisation sur un exemple.* Voici un scénario concret de problème de dépendances avec le système `apt`. Ce genre de problèmes peut cependant survenir avec la plupart des systèmes de gestion de dépendances. Si vous avez un système Linux basé sur Debian, vous pouvez même tester ce scénario en ajoutant les deux lignes suivantes au fichier `/etc/apt/sources.list` avant de lancer `sudo apt-get update` :

```
/etc/apt/sources.list
```

```
deb [trusted=yes] https://www.irif.fr/~schmitz/teach/2020_lo5/debs ./
deb [trusted=yes] https://www.irif.fr/~schmitz/teach/2020_lo5/debs-old ./
```

Un paquet logiciel Debian contient des méta-informations ; voici par exemple les informations du paquet `foo` disponible aux adresses ci-dessus :

```
apt-cache show foo
```

```
Package: foo
Version: 1.0
Architecture: all
Maintainer: anon
Depends: bar (>= 2.1), baz | qux
Filename: ./foo-1.0.deb
Size: 704
MD5sum: b898166d798077e84317686d66d259e5
SHA1: e4c056543faf48b4ba97fd2b113fd05397ea8a7d
SHA256: 591489045bf2ce5bc7c5d09cb3e9dd6416939ee23a38f4cd3ecba80889d717f7
Description: dummy foo-1.0 package
Description-md5: c777289cc850cccf5b3b07c9c31902f2
```

Ce qui nous intéresse est le nom du paquet (`foo`), sa version (`1.0`), et ses dépendances (`bar (>= 2.1)`, `baz | qux`) : le paquet `foo` dans sa version `1.0` dépend du paquet `bar` dans une version supérieure ou égale à `2.1`, et soit du paquet `baz` soit du paquet `qux`, sans contrainte de version.

À noter qu'un même paquet peut être disponible en plusieurs versions :

```
apt-cache show quxx
```

```
Package: quxx
Version: 1.4
Architecture: all
Maintainer: anon
Conflicts: quz
Filename: ./quxx-1.4.deb
Size: 664
MD5sum: ce56ed662469facfc2af728e75262849
SHA1: f1a91faf93a68b8eb626032607d148b494731124
```

```

SHA256: b7c4498b4b16c84e9315cf2ecea613ec11d943564ff2ef7d0c8b762fe10eced2
Description: dummy quxx-1.4 package
Description-md5: 0d88c6b67d64002702518fb93a4ebfee

Package: quxx
Version: 1.3
Architecture: all
Maintainer: anon
Filename: ./quxx-1.3.deb
Size: 680
MD5sum: 3553398d7d504fd1993c719436709f68
SHA1: c3bac75a683808f515656eaa3cf6460d71ab933b
SHA256: f58ac7d4ad72bf47b2ed4afaf97f9770f048076cf17ef53dae523cd8d66ac25a
Description: dummy quxx-1.3 package
Description-md5: 94ff5e3c00228e821c8163972d5fae10

```

Ici, quxx existe en version 1.3 et en version 1.4. Ces paquets n'ont pas de dépendances, mais quxx version 1.4 est en conflit avec quz : il ne peut pas être installé en même temps qu'aucune version de quz. Notons que les dépendances et conflits peuvent varier d'une version à l'autre d'un même paquet. La figure 8.13 résume les dépendances (en orange) et conflits (en rouge pointillé) entre les paquets de notre scénario.

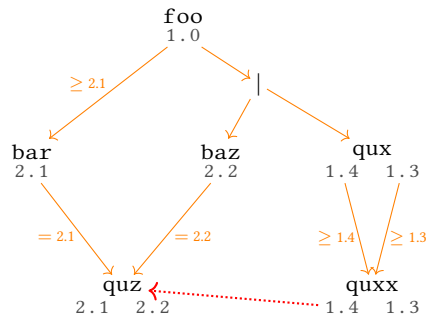


FIGURE 8.13. Les dépendances (en orange) et conflits (en rouge) entre paquets.

Si on essaie d'installer foo, on obtient un message d'erreur qui nous renseigne assez peu :

```
sudo apt-get install --dry-run foo
```

```
The following packages have unmet dependencies:
```

```
foo : Depends: bar (>= 2.1) but it is not going to be installed
```

```
E: Unable to correct problems, you have held broken packages.
```

Si on analyse la situation à l'aide du graphe de la figure 8.13, on peut voir que, pour pouvoir installer foo, on doit installer bar, et que ce dernier exige l'installation de quz dans sa version 2.1. On doit aussi installer baz ou qux. Le paquet baz exige l'installation de quz dans sa version 2.2 : ceci est incompatible avec quz-2.1 qui est une version différente du même paquet, donc on ne peut pas installer bar et baz en même temps. Il reste donc qux, mais on voit qu'il dépend de quxx qui dans sa version 1.4 est en conflit avec quz, donc on ne peut pas installer bar et qux-1.4 en même temps. Ceci explique le message d'erreur d'apt. Il y aurait cependant une solution si on se permettait d'installer des paquets dans

des versions qui ne sont pas les plus récentes : installer `bar-2.1`, `quz-2.1`, `qux-1.3` et `quxx-1.3`.

Voyons maintenant comment modéliser notre scénario en logique propositionnelle. Formellement, on dispose d'un ensemble de paquets versionnés

$$\mathcal{D} \stackrel{\text{def}}{=} \{\text{foo-1.0}, \text{bar-2.1}, \text{baz-2.2}, \text{qux-1.3}, \text{qux-1.4}, \text{quxx-1.3}, \text{quxx-1.4}, \text{quz-2.1}, \text{quz-2.2}\}$$

et on cherche une *solution*, c'est-à-dire un sous-ensemble $\mathcal{E} \subseteq \mathcal{D}$ tel que \mathcal{E} contienne `foo-1.0` le paquet que nous voulons installer, et tel que les dépendances et conflits de la figure 8.13 soient respectés.

Choix des propositions. Nous allons utiliser des propositions Q_p pour chaque paquet versionné $p \in \mathcal{D}$. Une interprétation I de ces propositions définit alors le sous-ensemble $\{p \mid Q_p^I = \mathbf{V}\} \subseteq \mathcal{D}$. Les contraintes que nous allons écrire assureront ensuite que `foo-1.0` appartienne au sous-ensemble ainsi défini et que les dépendances et conflits soient respectés.

Voici le préambule d'un fichier au format DIMACS où on a indiqué en commentaire quel entier strictement positif est associé à chaque paquet versionné $p \in \mathcal{D}$.

```
dependances.cnf
```

```
c Table des propositions : une proposition par version de paquet
c foo-1.0 bar-2.1 baz-2.2 qux-1.3 qux-1.4 quz-2.1 quz-2.2 quxx-1.3 quxx-1.4
c 1 2 3 4 5 6 7 8 9
c
p cnf 9 12
```

Nous allons maintenant écrire une formule propositionnelle qui sera satisfaite par une interprétation I si et seulement si I définit une solution.

Le paquet `foo-1.0` appartient à la solution. On garantit cela à l'aide de la formule $Q_{\text{foo-1.0}}$. En format DIMACS :

```
dependances.cnf
```

```
c on souhaite installer foo-1.0
1 0
```

Dépendances. Dans les cas les plus simples, pour chaque dépendance $p \rightarrow p'$ indiquée en orange dans la figure 8.13, on va ajouter une contrainte de la forme $Q_p \rightarrow Q_{p'}$: si p est installé, alors on doit aussi installer p' . Cela donne par exemple la formule $Q_{\text{foo-1.0}} \rightarrow Q_{\text{bar-2.1}}$.

De manière générale, une dépendance peut mener à une *disjonction* de paquets versionnés, et la contrainte logique correspondante est alors une implication d'une disjonction des propositions correspondantes ; par exemple $Q_{\text{foo-1.0}} \rightarrow (Q_{\text{baz-2.2}} \vee Q_{\text{qux-1.3}} \vee Q_{\text{qux-1.4}})$.

Ces deux contraintes peuvent être mises sous forme normale conjonctive comme deux clauses $\neg Q_{\text{foo-1.0}} \vee Q_{\text{bar-2.1}}$ et $\neg Q_{\text{foo-1.0}} \vee Q_{\text{baz-2.2}} \vee Q_{\text{qux-1.3}} \vee Q_{\text{qux-1.4}}$; voici leur traduction au format DIMACS :

```
dependances.cnf
```

```
c foo-1.0 dépend de : bar (>= 2.1), baz | qux
-1 2 0
-1 3 4 5 0
```

Les autres dépendances sont traitées de manière similaire.

Conflits déclarés. Il y a un conflit déclaré dans la figure 8.13 : quxx-1.4 est en conflit avec quz. On peut modéliser cela par la formule $Q_{\text{quxx-1.4}} \rightarrow \neg(Q_{\text{quz-2.1}} \vee Q_{\text{quz-2.2}})$: si on installe quxx-1.4, alors aucune version de quz ne peut être installée. Cette formule est logiquement équivalente à la formule $(\neg Q_{\text{quxx-1.4}} \vee \neg Q_{\text{quz-2.1}}) \wedge (\neg Q_{\text{quxx-1.4}} \vee \neg Q_{\text{quz-2.2}})$ sous forme normale conjonctive, et voici sa traduction au format DIMACS :

```
dependances.cnf
c quxx-1.4 est en conflit avec : quz
-9 -6 0
-9 -7 0
```

Conflits entre versions d'un même paquet. Enfin, il faut ajouter les conflits implicites : deux versions différentes d'un même paquet ne peuvent pas coexister. Par exemple, pour qux, cela revient à la formule propositionnelle $\neg(Q_{\text{qux-1.3}} \wedge Q_{\text{qux-1.4}})$, qui est logiquement équivalente à $\neg Q_{\text{qux-1.3}} \vee \neg Q_{\text{qux-1.4}}$ sous forme normale conjonctive ; voici sa traduction au format DIMACS :

```
dependances.cnf
c on ne peut pas avoir deux versions de qux
-4 -5 0
```

Des contraintes similaires doivent être écrites pour quxx et quz.

Si on appelle un solveur SAT comme MINISAT sur le fichier DIMACS complet, celui-ci trouve un modèle


```
dependances.modele
SAT
1 2 -3 4 -5 6 -7 8 -9 0
```

Cette interprétation satisfait nos dépendances et conflits et correspond à la solution

$$\{\text{foo-1.0}, \text{bar-2.1}, \text{qux-1.3}, \text{quxx-1.3}, \text{quz-2.1}\}.$$

La commande `sudo apt-get install quxx=1.3 qux=1.3 quz=2.1 bar=2.1 foo` fonctionne maintenant et permet d'installer foo.

8.5. Application : coloration de graphe. Les solveurs SAT sont particulièrement utiles pour des problèmes combinatoires pour lesquels on ne connaît pas d'algorithme efficace dans le pire des cas – et on soupçonne qu'il n'existe pas de tels algorithmes. Il s'avère en effet que les solveurs SAT permettent de résoudre ces problèmes dans les cas *qui apparaissent en pratique*.

8.5.1.  *Modélisation sur un exemple.* Notre problème est le suivant : est-il possible de colorier la carte de l'Australie (voir la carte⁴) avec seulement trois couleurs, disons rouge, vert et bleu, de telle sorte que deux territoires adjacents aient des couleurs différentes ? Ce problème est en réalité un problème de coloration d'un graphe non orienté comme illustré à droite de la figure 8.14 : peut-on associer une couleur à chaque sommet du graphe, de telle

4. Auteur de la carte à gauche de la figure 8.14 : Lokal_Profil, licence [CC BY-SA 3.0], via *Wikimedia Commons*. Les territoires sont : *Western Australia* (WA), *Northern Territory* (NT), *South Australia* (SA), *Queensland* (QLD), *New South Wales* (NSW), *Victoria* (VIC) et *Tasmania* (TAS) ; on a ignoré le petit territoire de la capitale.

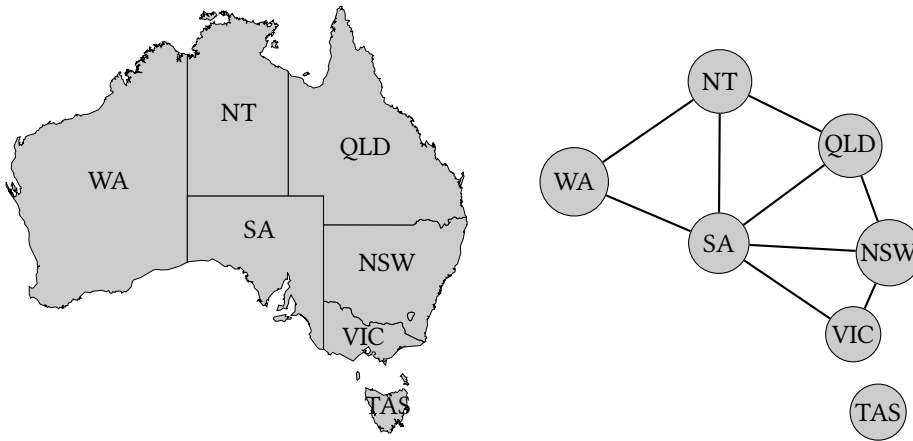


FIGURE 8.14. La carte des territoires de l'Australie⁴ et le graphe associé.

sorte que deux sommets adjacents n'aient pas la même couleur? La réponse est « oui » et une telle coloration est donnée dans la figure 8.15.

Notre objectif est cependant de trouver automatiquement une telle solution pour n'importe quel graphe fini non orienté. Voyons comment procéder pour notre exemple.

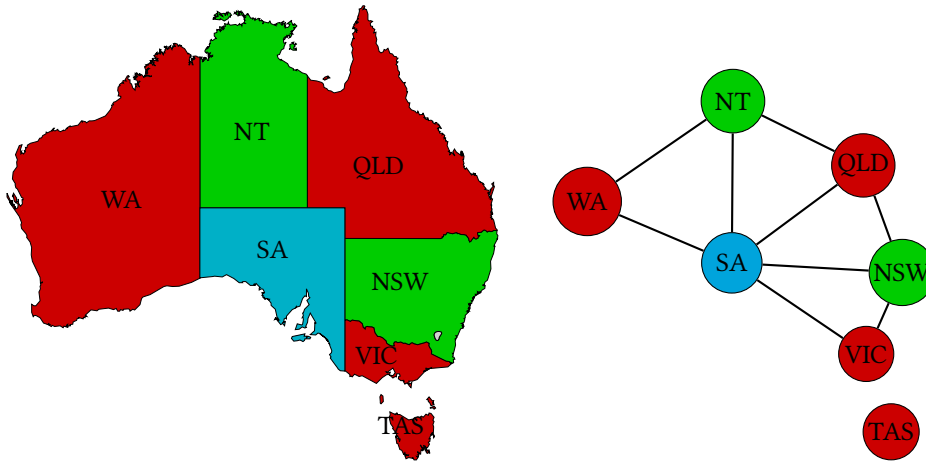


FIGURE 8.15. Une coloration de la carte des territoires de l'Australie⁴ et du graphe associé.

Choix des propositions. Nous travaillons avec des propositions $P_{v,c}$ où v est un sommet du graphe, c'est-à-dire un territoire dans $\{WA, NT, SA, QLD, NSW, VIC, TAS\}$ et c est une couleur dans $C \stackrel{\text{def}}{=} \{R, V, B\}$. Cela signifie qu'une interprétation I de ces propositions

décriera une *relation* incluse dans $V \times C$: pour chaque paire (v, c) , I va indiquer si le sommet v du graphe est colorié par la couleur c .

Voici le préambule d'un fichier au format DIMACS où on a indiqué en commentaire quel entier strictement positif est associé à chacune des paires $(v, c) \in V \times C$.

```
coloration.cnf
c table des propositions
c WA,R WA,V WA,B NT,R NT,V NT,V SA,R SA,V SA,B QLD,R QLD,V QLD,B
c 1 2 3 4 5 6 7 8 9 10 11 12
c NSW,R NSW,V NSW,B VIC,R VIC,V VIC,B TAS,R TAS,V TAS,B
c 13 14 15 16 17 18 19 20 21
p cnf 21 55
```

Nous allons maintenant construire une formule propositionnelle qui sera satisfiable si et seulement s'il existe une coloration du graphe. Toutes les interprétations I des propositions de la forme $P_{v,c}$ pour $v \in \{WA, NT, SA, QLD, NSW, VIC, TAS\}$ et $c \in \{R, V, B\}$ ne sont pas des colorations. Plusieurs conditions doivent en effet être remplies.

Au moins une couleur par sommet. Nous ne devons pas laisser un territoire de l'Australie non colorié. Cela correspond à vérifier que pour chaque territoire $v \in \{WA, NT, SA, QLD, NSW, VIC, TAS\}$, au moins l'une des propositions parmi $P_{v,R}$, $P_{v,V}$ et $P_{v,B}$ est vraie dans I , soit la formule propositionnelle $P_{v,R} \vee P_{v,V} \vee P_{v,B}$. Voici les clauses correspondantes au format DIMACS.

```
coloration.cnf
c au moins une couleur par sommet
1 2 3 0
4 5 6 0
7 8 9 0
10 11 12 0
13 14 15 0
16 17 18 0
19 20 21 0
```

Au plus une couleur par sommet. En effet, par exemple, le territoire de *New South Wales* ne peut pas être colorié à la fois en rouge et en vert. Cela correspond à vérifier que, pour chaque territoire $v \in \{WA, NT, SA, QLD, NSW, VIC, TAS\}$ et pour chaque paire de couleurs distinctes $c \neq c'$ issues de $\{R, V, B\}$, on n'ait pas à la fois $P_{v,c}$ et $P_{v,c'}$ vraies dans I . Cela correspond à la formule propositionnelle $\neg(P_{v,c} \wedge P_{v,c'})$, qui est équivalente à la clause $\neg P_{v,c} \vee \neg P_{v,c'}$. Voici les clauses correspondantes au format DIMACS pour WA, NT et SA.

```
coloration.cnf
c au plus une couleur par sommet
-1 -2 0
-1 -3 0
-2 -3 0
-4 -5 0
-4 -6 0
-5 -6 0
-7 -8 0
-7 -9 0
-8 -9 0
```

Avec ces deux types de clauses combinées, on garantit en fait qu'il existe une *fonction* entre les sommets du graphe et les couleurs.

Couleurs distinctes pour sommets adjacents. Enfin, nous devons vérifier que pour toute paire $\{u, v\}$ de territoires adjacents, leurs couleurs associées sont distinctes, c'est-à-dire que I vérifie $\neg(P_{u,c} \wedge P_{v,c})$ pour tout arête (u, v) et toute couleur c ; cela s'écrit sous forme clause comme $\neg P_{u,c} \vee \neg P_{v,c}$. Par exemple, pour les arêtes (WA, NT), (WA, SA) et (NT, SA), on aura au format DIMACS :

```
coloration.cnf
c pas la même couleur sur deux sommets adjacents
c arête (WA,NT)
-1 -4 0
-2 -5 0
-3 -6 0
c arête (WA,SA)
-1 -7 0
-2 -8 0
-3 -9 0
c arête (NT,SA)
-4 -7 0
-5 -8 0
-6 -9 0
```

8.5.2. *Réduction dans le cas général.* En général, une k -coloration d'un graphe non orienté $G = (V, E)$ est une fonction $c: V \rightarrow \{1, \dots, k\}$ telle que, pour toute arête $\{v, v'\}$ de E , $c(v) \neq c(v')$. Le problème de COLORATION DE GRAPHE prend en entrée un graphe fini G et un nombre de couleurs k et demande si une telle k -coloration existe.

On peut légèrement reformuler le problème de COLORATION DE GRAPHE comme celui de l'existence d'une relation binaire $R \subseteq V \times \{1, \dots, k\}$ telle que

- (1) R est le graphe d'une fonction de type $V \rightarrow \{1, \dots, k\}$ et
- (2) pour toute arête $\{v, v'\} \in E$ et couleur $1 \leq i \leq k$, $(v, i) \notin R$ ou $(v', i) \notin R$.

On vient de voir dans le cas particulier de la section 8.5.1 comment réduire ce problème à SAT. L'idée plus généralement quand on recherche l'existence d'une relation binaire $R \subseteq A \times B$ pour deux ensembles finis A et B est de travailler sur les propositions $P_{a,b}$ pour $a \in A$ et $b \in B$; une interprétation I de ces propositions $P_{a,b}$ définit en effet une relation

$$R_I \stackrel{\text{def}}{=} \{(a, b) \in A \times B \mid P_{a,b}^I = \mathbf{V}\}. \quad (8.1)$$

Dans le cas de l'exemple de modélisation de la section 8.5.1, on obtient le schéma de la figure 8.16; dans le cas général du problème de COLORATION DE GRAPHE, $A = V$ et $B = \{1, \dots, k\}$.

8.5.2(1). *Graphe d'une fonction.* On peut exprimer diverses contraintes sur une relation R_I définie par une interprétation I comme dans l'équation (8.1). En particulier, une telle relation R_I est le graphe d'une fonction si et seulement si elle est totale et fonctionnelle, c'est-à-dire dans le cas de k -coloration qui nous intéresse, si et seulement si elle satisfait

🗨 *Le problème COLORATION DE GRAPHE est NP-complet, même pour $k \geq 3$ fixé (KARP, 1972; GAREY et JOHNSON, 1979, [GT4]).*

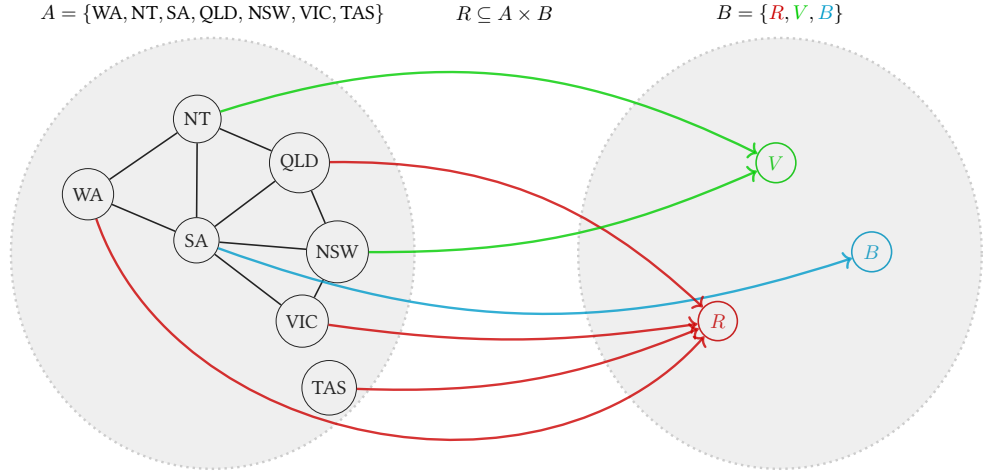


FIGURE 8.16. Une $\{R, V, B\}$ -coloration de graphe vue comme l'existence d'une relation $R \subseteq V \times \{R, V, B\}$.

l'ensemble $\{\chi_v \mid v \in V\}$ de formules définies par

$$\chi_v \stackrel{\text{def}}{=} \left(\bigvee_{1 \leq i \leq k} P_{v,i} \right) \wedge \left(\bigwedge_{1 \leq i < j \leq k} (\neg P_{v,i} \vee \neg P_{v,j}) \right). \quad (\text{totalité et fonctionnalité})$$

8.5.2.(2). *k-coloration*. La deuxième contrainte pour que la relation R_I soit bien le graphe d'une k -coloration s'exprime quant à elle par l'ensemble $\{\chi_{\{v,v'\}} \mid \{v,v'\} \in E\}$ de formules définies par

$$\chi_{\{v,v'\}} \stackrel{\text{def}}{=} \bigwedge_{1 \leq i \leq k} (\neg P_{v,i} \vee \neg P_{v',i}). \quad (\text{coloration})$$

En combinant ces deux ensembles de formules pour un graphe $G = (V, E)$, on a que $I \models \varphi_{G,k}$ où $\varphi_{G,k} \stackrel{\text{def}}{=} (\bigwedge_{v \in V} \chi_v) \wedge (\bigwedge_{\{v,v'\} \in E} \chi_{\{v,v'\}})$ si et seulement si la relation R_I satisfait les conditions (1) et (2) de l'existence d'une k -coloration : en d'autres termes, il existe une k -coloration de G si et seulement si la formule $\varphi_{G,k}$ est satisfiable.

On peut noter que cette formule peut être construite en espace logarithmique à partir de la description de G et k , et est même sous forme k -CNF. On vient donc de décrire une réduction en espace logarithmique de COLORATION DE GRAPHE vers SAT. Ce n'est certainement pas la démonstration la plus simple du fait que COLORATION DE GRAPHE soit dans NP ; l'intérêt de cette réduction réside dans le fait que l'on dispose de solveurs SAT assez performants en pratique pour ce type d'entrées, et donc aussi aptes à résoudre le problème de COLORATION DE GRAPHE.

8.5.3. * *Théorème de DE BRUIJN et ERDŐS*. La réduction de COLORATION DE GRAPHE à SAT esquissée juste au-dessus peut être combinée au théorème de compacité pour fournir une démonstration du théorème suivant sur les colorations de graphes infinis.

Théorème 8.13 (DE BRUIJN et ERDŐS). *Un graphe infini a une k -coloration si et seulement si chacun de ses sous-graphes finis a une k -coloration.*

Démonstration. Si un graphe infini $G = (V, E)$ a une k -coloration $c: V \rightarrow \{1, \dots, k\}$, alors pour tout sous-graphe fini (V', E') avec $V' \subseteq V$ et $E' \subseteq E$, la restriction de la fonction c à V' est une k -coloration.

Inversement, supposons que tout sous-graphe fini (V', E') de $G = (V, E)$ avec $V' \subseteq V$ et $E' \subseteq E$ a une k -coloration. On définit l'ensemble de formules propositionnelles

$$S_G \stackrel{\text{def}}{=} \{\chi_v \mid v \in V\} \cup \{\chi_{\{v,v'\}} \mid \{v, v'\} \in E\}.$$

Soit $F \subseteq S_G$ un sous-ensemble fini de S_G . Alors il existe $V' \subseteq V$ et $E' \subseteq E$ deux ensembles finis tels que $F = \{\chi_v \mid v \in V'\} \cup \{\chi_{\{v,v'\}} \mid \{v, v'\} \in E'\}$, et puisque le sous-graphe fini (V', E') a une k -coloration, cet ensemble F est satisfiable. Par le théorème 6.13 de compacité, comme on vient de montrer que tout sous-ensemble fini de S_G est satisfiable, l'ensemble est satisfiable. Mais alors $G = (V, E)$ a une k -coloration. \square

Partie 3

Dédution naturelle

Programme officiel. Il s'agit de présenter les preuves comme permettant de pallier deux problèmes de la présentation précédente du calcul propositionnel : nature exponentielle de la vérification d'une tautologie, faible lien avec les preuves mathématiques.

Il ne s'agit, en revanche, que d'introduire la notion d'arbre de preuve. La déduction naturelle est présentée comme un jeu de règles d'inférence simple permettant de faire un calcul plus efficace que l'étude de la table de vérité. Toute technicité dans les preuves dans ce système est à proscrire.

Notions	Commentaires
Règle d'inférence, dérivation.	Notation \vdash . Séquent $H_1, \dots, H_n \vdash C$. On présente des exemples tels que le <i>modus ponens</i> ($p, p \rightarrow q \vdash q$) ou le syllogisme <i>barbara</i> ($p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$).
Définition inductive d'un arbre de preuve.	On présente des exemples utilisant les règles précédentes.
Règles d'introduction et d'élimination de la déduction naturelle pour les formules propositionnelles. Correction de la déduction naturelle pour les formules propositionnelles.	On présente les règles pour \wedge, \vee, \neg et \rightarrow . On écrit de petits exemples d'arbre de preuves (par exemple $\vdash (p \rightarrow q) \rightarrow \neg(p \wedge \neg q)$, etc.).
Règles d'introduction et d'élimination pour les quantificateurs universels et existentiels.	On motive ces règles par une approche sémantique intuitive.

Mise en œuvre Il ne s'agit pas d'implémenter ces règles mais plutôt d'être capable d'écrire de petites preuves dans ce système. On peut également présenter d'autres utilisations de règles d'inférences pour raisonner.

Le problème de *validité* est le problème de décision suivant.

Problème (VALIDITÉ).

instance : une formule propositionnelle φ

question : φ est-elle valide ?

Une façon de résoudre ce problème nous est offerte par la dualité entre satisfiabilité et validité (voir la propriété 5.11) : φ est valide si et seulement si $\neg\varphi$ n'est pas satisfiable. On peut donc résoudre VALIDITÉ à l'aide d'un solveur SAT. Alternativement, on pourrait calculer la table de vérité de φ et vérifier qu'elle ne contient que la valeur \mathbf{V} pour toute interprétation.

Cependant, on souhaiterait aussi, dans le cas où φ est valide, avoir un *certificat* de validité, qui « explique » pourquoi la formule propositionnelle est valide ; on voudrait de plus pouvoir

aisément vérifier, quand on dispose d'un tel certificat, que la formule propositionnelle était bien valide. Les solveurs SAT récents fournissent des certificats d'insatisfiabilité, tandis que la table de vérité elle-même est un certificat de validité, mais dans ces deux approches ces certificats n'*expliquent* pas grand-chose.

☞ On soupçonne que des certificats de taille polynomiale et vérifiables en temps polynomiale n'existent pas pour VALIDITÉ, car cela impliquerait $NP = coNP$. Cette question de la « complexité des preuves » a été initiée par COOK et RECKHOW; remarquons au passage que $NP \neq coNP$ impliquerait $P \neq NP$.

L'approche alternative que nous allons explorer dans cette partie est plutôt de résoudre le problème de validité directement, à l'aide d'un *système de preuve*, c'est-à-dire de règles de déduction qui garantissent la validité. L'intérêt ici est que la preuve elle-même, c'est-à-dire l'arbre de dérivation dans le système de preuve, fait office de certificat. Un tel certificat est très facile à vérifier (et c'est d'ailleurs ce que font les assistants de preuve). Comme les certificats d'insatisfiabilité des solveurs SAT et la table de vérité de la formule, cette dérivation peut être potentiellement de taille exponentielle en la taille de φ . Mais contrairement aux ceux-ci, la dérivation dans le système de preuve a une valeur explicative, car elle reflète (de manière très formelle et détaillée) la façon dont on pourrait organiser une preuve mathématique de validité (voir les exemples 9.1 à 9.3).

9. DÉDUCTION NATURELLE CLASSIQUE PROPOSITIONNELLE

Résumé. La *dédution naturelle* est un système de déduction qui travaille sur des *séquents* de la forme $\Gamma \vdash \varphi$, où Γ est un ensemble de formules propositionnelles et φ une formule propositionnelle. Outre la règle d'axiome et la règle d'absurdité classique, ce système de déduction comporte des règles d'*introduction* et d'*élimination* pour chacun des connecteurs logiques \neg, \vee, \wedge et \rightarrow . Un séquent $\Gamma \vdash \varphi$ pour lequel il existe une dérivation dans le système de déduction est dit *prouvable*, ce qui est noté « $\Gamma \vdash_{\text{NK}_0} \varphi$ ».

Un séquent $\Gamma \vdash \varphi$ est *valide* si φ est une conséquence logique de Γ , c'est-à-dire si $\Gamma \models \varphi$. Par le théorème 9.10 de correction propositionnelle de la déduction naturelle, si $\Gamma \vdash \varphi$ est prouvable, alors il est valide. Inversement, par le théorème 9.11 de complétude propositionnelle de la déduction naturelle, si $\Gamma \vdash \varphi$ est valide, alors il est prouvable. La déduction naturelle fournit donc un moyen d'établir la validité d'une formule propositionnelle exclusivement par des manipulations syntaxiques.

Les règles de la déduction naturelle peuvent être implémentées au sein d'un *assistant de preuve* tel que Coq, qui se charge de vérifier que la dérivation est bien correcte (section 9.4).

En déduction naturelle, on va travailler avec une syntaxe quelque peu étendue pour les formules propositionnelles :

$$\varphi ::= \perp \mid P \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \varphi \rightarrow \psi$$

où P est tiré de \mathcal{P}_0 . Il existe plusieurs manières de présenter ce système de preuve, et nous allons en voir une présentation à l'aide de *séquents*, qui sont dans cette section et la prochaine des paires notées « $\Gamma \vdash \varphi$ » et constituées d'un ensemble fini Γ de formules propositionnelles (noté comme une liste de formules séparées par des virgules) et d'une formule propositionnelle φ . La virgule dénote l'union entre ensembles de formules dans des séquents : par exemple, si Γ et Δ sont des ensembles de formules et φ et ψ sont des formules, alors on écrira $\Gamma, \Delta, \varphi \vdash \psi$ plutôt que $\Gamma \cup \Delta \cup \{\varphi\} \vdash \psi$.

Les règles de la déduction naturelle sont essentiellement scindées en deux groupes pour chaque connecteur logique. Les règles dites d'*introduction* permettent de faire apparaître un connecteur logique dans la partie droite d'un séquent, tandis que les règles d'*élimination* permettent de le faire disparaître.

On dit qu'un séquent $\Gamma \vdash \varphi$ est *prouvable* en déduction naturelle, noté $\Gamma \vdash_{\text{NK}_0} \varphi$, s'il existe une dérivation du séquent à l'aide des règles de la figure 9.1.

■ Voir (DUPARC, 2015, sec. I.3.3), (DAVID, NOUR et RAFFALLI, 2003, sec. 1.3.3), (GOUBAULT-LARRECQ et MACKIE, 1997, sec. 2.3.2), (MANCOSU, GALVAN et ZACH, 2021, chap. 3), (TROELSTRA et SCHWICHTENBERG, 2000, sec. 2.1.8).

☞ La déduction naturelle possède aussi une version minimale et une version intuitionniste, c.f. (DUPARC, 2015, rem. 138), (DAVID, NOUR et RAFFALLI, 2003, chap. 4), (GOUBAULT-LARRECQ et MACKIE, 1997, sec. 3.2). C'est d'ailleurs pour ces variantes et leurs relations avec le lambda-calcul via l'isomorphisme de CURRY-HOWARD que la déduction naturelle trouve tout son intérêt.

$$\begin{array}{c}
\frac{}{\Gamma, \varphi \vdash \varphi} \text{ (ax)} \\
\frac{\Gamma, \varphi \vdash \perp}{\Gamma \vdash \neg \varphi} \text{ (}\neg\text{i)} \\
\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \text{ (}\vee\text{i}_g) \\
\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi} \text{ (}\wedge\text{i)} \\
\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi} \text{ (}\rightarrow\text{i)} \\
\frac{\Gamma, \neg \varphi \vdash \perp}{\Gamma \vdash \varphi} \text{ (}\perp\text{c)} \\
\frac{\Gamma \vdash \neg \varphi \quad \Gamma \vdash \varphi}{\Gamma \vdash \perp} \text{ (}\neg\text{e)} \\
\frac{\Gamma \vdash \varphi \vee \psi \quad \Gamma, \varphi \vdash \delta \quad \Gamma, \psi \vdash \delta}{\Gamma \vdash \delta} \text{ (}\vee\text{e)} \\
\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi} \text{ (}\wedge\text{e}_g) \\
\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi} \text{ (}\wedge\text{e}_d) \\
\frac{\Gamma \vdash \varphi \rightarrow \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi} \text{ (}\rightarrow\text{e)}
\end{array}$$

FIGURE 9.1. Déduction naturelle classique propositionnelle.

9.1. Exemples de dérivations en déduction naturelle. Les dérivations en déduction naturelle ont souvent une ressemblance avec l'organisation d'une démonstration mathématique. Les trois premiers exemples ci-dessous fournissent de telles démonstrations annotées par des règles de déduction naturelle.

Exemple 9.1 (modus ponens). Voici une dérivation en déduction naturelle qui montre $P, P \rightarrow Q \vdash_{\text{NK}_0} Q$.

$$\frac{\frac{P, P \rightarrow Q \vdash P \rightarrow Q}{} \text{ (ax)} \quad \frac{P, P \rightarrow Q \vdash P}{} \text{ (ax)}}{P, P \rightarrow Q \vdash Q} \text{ (}\rightarrow\text{e)}$$

Voyons comment on aurait pu montrer que Q est une conséquence logique de l'ensemble $\{P, P \rightarrow Q\}$ par un raisonnement sur les interprétations, annoté avec les règles de déduction naturelle que nous venons d'utiliser.

Soit I une interprétation telle que $I \models P$ (ax) et $I \models P \rightarrow Q$ (ax). Alors $I \models Q$ (\rightarrow e).

Exemple 9.2 (barbara). Voici une dérivation en déduction naturelle qui montre $P \rightarrow Q, Q \rightarrow R \vdash_{\text{NK}_0} P \rightarrow R$.

$$\frac{\frac{P \rightarrow Q, Q \rightarrow R, P \vdash Q \rightarrow R}{} \text{ (ax)} \quad \frac{\frac{P \rightarrow Q, Q \rightarrow R, P \vdash P \rightarrow Q}{} \text{ (ax)} \quad \frac{P \rightarrow Q, Q \rightarrow R, P \vdash P}{} \text{ (ax)}}{P \rightarrow Q, Q \rightarrow R, P \vdash Q} \text{ (}\rightarrow\text{e)}}{\frac{P \rightarrow Q, Q \rightarrow R, P \vdash R}{P \rightarrow Q, Q \rightarrow R \vdash P \rightarrow R} \text{ (}\rightarrow\text{i)}}$$

Voyons comment on aurait pu montrer que $P \rightarrow R$ est une conséquence logique de l'ensemble $\{P \rightarrow Q, Q \rightarrow R\}$ par un raisonnement sur les interprétations, annoté avec les règles de déduction naturelle que nous venons d'utiliser.

Soit I une interprétation telle que $I \models P \rightarrow Q$ (ax) et $I \models Q \rightarrow R$ (ax). Pour montrer que $I \models P \rightarrow R$, on va supposer de plus que $I \models P$ (ax). Alors puisque $I \models P \rightarrow Q$ et $I \models P$, $I \models Q$ (\rightarrow e). Comme de plus $I \models Q \rightarrow R$, on a aussi $I \models R$ (\rightarrow e). On vient de montrer que $\{P \rightarrow Q, Q \rightarrow R, P\} \models R$, et par le lemme 6.2 de déduction, on en déduit $\{P \rightarrow Q, Q \rightarrow R\} \models P \rightarrow R$ (\rightarrow i).

Exemple 9.3. Voici une dérivation en déduction naturelle qui montre que $\vdash_{\text{NK}_0} (P \rightarrow Q) \rightarrow \neg(P \wedge \neg Q)$, où on a noté $\Gamma \stackrel{\text{def}}{=} P \rightarrow Q, P \wedge \neg Q$.

$$\frac{\frac{\frac{\Gamma \vdash P \rightarrow Q}{\Gamma \vdash Q} \text{ (ax)}}{\Gamma \vdash Q} \quad \frac{\frac{\frac{\Gamma \vdash P \wedge \neg Q}{\Gamma \vdash P} \text{ (ax)}}{\Gamma \vdash P} \text{ (\wedge_e)} \quad \frac{\frac{\Gamma \vdash P \wedge \neg Q}{\Gamma \vdash \neg Q} \text{ (ax)}}{\Gamma \vdash \neg Q} \text{ (\wedge_d)}}{\Gamma \vdash \perp} \text{ (-e)}}{\Gamma \vdash \perp} \text{ (-i)}}{\frac{P \rightarrow Q \vdash \neg(P \wedge \neg Q)}{\vdash (P \rightarrow Q) \rightarrow \neg(P \wedge \neg Q)} \text{ (-i)}} \text{ (-i)}$$

Voyons comment démontrer la validité de la formule $(P \rightarrow Q) \rightarrow \neg(P \wedge \neg Q)$ par un raisonnement sur les interprétations. Soit I une interprétation. On va montrer que, si $I \models P \rightarrow Q$ (ax), alors $I \models \neg(P \wedge \neg Q)$, ce qui permettra de conclure par le lemme 6.2 de déduction que $\models (P \rightarrow Q) \rightarrow \neg(P \wedge \neg Q)$ (\rightarrow i).

Supposons par l'absurde que $I \models P \wedge \neg Q$ (ax) et montrons que l'on aboutit à une contradiction (\rightarrow i). On va montrer pour cela que $I \models Q$ et $I \models \neg Q$ (\rightarrow e) :

- d'une part, comme $I \models P \wedge \neg Q$, en particulier $I \models P$ (\wedge_e), et comme de plus $I \models P \rightarrow Q$, alors $I \models Q$ (\rightarrow_e);
- d'autre part, comme $I \models P \wedge \neg Q$, en particulier $I \models \neg Q$ (\wedge_d).

Plus généralement, le lien entre prouvabilité en déduction naturelle et conséquences logiques sera établi formellement en section 9.3 dans les théorèmes de correction et de complétude. La recherche de preuve en déduction naturelle n'est pas aisée. On verra en section 11 comment on peut automatiser ce processus en faisant un détour par un autre système de preuve plus adapté pour la recherche de preuve, mais pour des preuves « à la main » il faut s'entraîner sur des exemples.

Exemple 9.4 (double négation). Voici une dérivation en déduction naturelle qui montre $\neg\neg P \vdash_{\text{NK}_0} P$.

$$\frac{\frac{\frac{\neg\neg P, \neg P \vdash \neg\neg P}{\neg\neg P, \neg P \vdash \perp} \text{ (ax)}}{\neg\neg P, \neg P \vdash \perp} \text{ (-e)}}{\neg\neg P \vdash P} \text{ (\perp_c)}$$

Exemple 9.5 (loi de PEIRCE). Voici une dérivation en déduction naturelle qui montre que la loi de PEIRCE de l'exemple 5.10 est prouvable : $\vdash_{\text{NK}_0} ((P \rightarrow Q) \rightarrow P) \rightarrow Q$, où on a noté $\Gamma \stackrel{\text{def}}{=} (P \rightarrow Q) \rightarrow P$.

$$\frac{\frac{\frac{\frac{\frac{\Gamma, \neg P, P, \neg Q \vdash \neg P}{\Gamma, \neg P, P, \neg Q \vdash \perp} \text{ (ax)}}{\Gamma, \neg P, P, \neg Q \vdash \perp} \text{ (-e)}}{\Gamma, \neg P, P \vdash Q} \text{ (\wedge_e)} \quad \frac{\frac{\Gamma, \neg P, P, \neg Q \vdash \perp}{\Gamma, \neg P, P \vdash Q} \text{ (\perp_c)}}{\Gamma, \neg P \vdash P \rightarrow Q} \text{ (-i)}}{\Gamma, \neg P \vdash (P \rightarrow Q) \rightarrow P} \text{ (ax)}}{\Gamma, \neg P \vdash P} \text{ (-e)}}{\frac{\Gamma, \neg P \vdash \perp}{\Gamma \vdash P} \text{ (\perp_c)}} \text{ (-e)}}{\vdash ((P \rightarrow Q) \rightarrow P) \rightarrow P} \text{ (-i)}$$

Exemple 9.6 (dualité de DE MORGAN pour \vee). Voici une dérivation en déduction naturelle qui montre $\neg(\varphi \vee \psi) \vdash_{\text{NK}_0} \neg\varphi \wedge \neg\psi$, où on a noté $\Gamma \stackrel{\text{def}}{=} \neg(\varphi \vee \psi)$.

$$\frac{\frac{\frac{\Gamma, \varphi \vdash \neg(\varphi \vee \psi)}{\Gamma, \varphi \vdash \neg\varphi} \text{ (ax)} \quad \frac{\frac{\Gamma, \varphi \vdash \varphi} \text{ (ax)} \quad \frac{\Gamma, \varphi \vdash \varphi \vee \psi} \text{ (}\vee\text{id)}}{\Gamma, \varphi \vdash \neg\varphi} \text{ (}\neg\text{e)}}{\Gamma, \varphi \vdash \perp} \text{ (}\neg\text{i)}}{\Gamma \vdash \neg\varphi} \text{ (}\neg\text{i)}} \quad \frac{\frac{\frac{\Gamma, \psi \vdash \neg(\varphi \vee \psi)}{\Gamma, \psi \vdash \neg\psi} \text{ (ax)} \quad \frac{\frac{\Gamma, \psi \vdash \psi} \text{ (ax)} \quad \frac{\Gamma, \psi \vdash \varphi \vee \psi} \text{ (}\vee\text{id)}}{\Gamma, \psi \vdash \neg\psi} \text{ (}\neg\text{e)}}{\Gamma, \psi \vdash \perp} \text{ (}\neg\text{i)}}{\Gamma \vdash \neg\psi} \text{ (}\neg\text{i)}}}{\Gamma \vdash \neg\varphi \wedge \neg\psi} \text{ (}\wedge\text{i)}$$

Exemple 9.7 (dualité de DE MORGAN pour \wedge). Voici une dérivation en déduction naturelle qui montre $\neg(\varphi \wedge \psi) \vdash_{\text{NK}_0} \neg\varphi \vee \neg\psi$, où on a noté $\Gamma \stackrel{\text{def}}{=} \neg(\varphi \wedge \psi)$, $\neg(\neg\varphi \vee \neg\psi)$.

$$\frac{\frac{\frac{\frac{\Gamma, \varphi, \psi \vdash \neg(\varphi \wedge \psi)}{\Gamma, \varphi, \psi \vdash \neg\varphi} \text{ (ax)} \quad \frac{\frac{\Gamma, \varphi, \psi \vdash \varphi} \text{ (ax)} \quad \frac{\Gamma, \varphi, \psi \vdash \psi} \text{ (}\wedge\text{i)}}{\Gamma, \varphi, \psi \vdash \varphi \wedge \psi} \text{ (}\wedge\text{e)}}{\Gamma, \varphi, \psi \vdash \perp} \text{ (}\neg\text{i)}}{\Gamma, \varphi, \psi \vdash \neg(\neg\varphi \vee \neg\psi)} \text{ (ax)} \quad \frac{\frac{\frac{\Gamma, \varphi, \psi \vdash \perp} \text{ (}\neg\text{i)}}{\Gamma, \varphi \vdash \neg\psi} \text{ (}\neg\text{e)}}{\Gamma, \varphi \vdash \neg(\neg\varphi \vee \neg\psi)} \text{ (}\neg\text{e)}}{\Gamma, \varphi \vdash \neg\varphi} \text{ (}\neg\text{i)}}{\Gamma \vdash \neg(\neg\varphi \vee \neg\psi)} \text{ (ax)} \quad \frac{\frac{\Gamma, \varphi \vdash \perp} \text{ (}\neg\text{i)}}{\Gamma \vdash \neg\varphi} \text{ (}\neg\text{i)}}{\Gamma \vdash \perp} \text{ (}\neg\text{i)}}{\neg(\varphi \wedge \psi) \vdash \neg\varphi \vee \neg\psi} \text{ (}\perp\text{c)}$$

Exemple 9.8 (implication). Voici une dérivation en déduction naturelle qui montre $\neg\varphi \vee \psi \vdash_{\text{NK}_0} \varphi \rightarrow \psi$ où on a noté $\Gamma \stackrel{\text{def}}{=} \neg\varphi \vee \psi, \varphi$.

$$\frac{\frac{\frac{\Gamma, \neg\varphi, \neg\psi \vdash \neg\varphi} \text{ (ax)} \quad \frac{\Gamma, \neg\varphi, \neg\psi \vdash \varphi} \text{ (ax)}}{\Gamma, \neg\varphi, \neg\psi \vdash \perp} \text{ (}\perp\text{c)}}{\Gamma, \neg\varphi \vdash \psi} \text{ (}\neg\text{e)}}{\Gamma \vdash \neg\varphi \vee \psi} \text{ (ax)} \quad \frac{\Gamma, \psi \vdash \psi} \text{ (ax)}}{\Gamma \vdash \psi} \text{ (}\rightarrow\text{i)}}{\neg\varphi \vee \psi \vdash \varphi \rightarrow \psi} \text{ (}\rightarrow\text{e)}$$

Exemple 9.9 (décurryfication). Voici une dérivation en déduction naturelle qui montre $\varphi \rightarrow (\psi \rightarrow \delta) \vdash_{\text{NK}_0} (\varphi \wedge \psi) \rightarrow \delta$, où on a noté $\Gamma \stackrel{\text{def}}{=} \varphi \rightarrow (\psi \rightarrow \delta), \varphi \wedge \psi$.

$$\frac{\frac{\frac{\Gamma \vdash \varphi \rightarrow (\psi \rightarrow \delta)}{\Gamma \vdash \psi \rightarrow \delta} \text{ (ax)} \quad \frac{\frac{\Gamma \vdash \varphi \wedge \psi} \text{ (ax)} \quad \frac{\Gamma \vdash \varphi} \text{ (}\wedge\text{eg)}}{\Gamma \vdash \varphi} \text{ (}\rightarrow\text{e)}}{\Gamma \vdash \varphi \wedge \psi} \text{ (}\wedge\text{ed)}}{\Gamma \vdash \psi} \text{ (}\rightarrow\text{e)}}{\Gamma \vdash \delta} \text{ (}\rightarrow\text{i)}}{\varphi \rightarrow (\psi \rightarrow \delta) \vdash (\varphi \wedge \psi) \rightarrow \delta} \text{ (}\rightarrow\text{i)}$$

■ (DAVID, NOUR et RAFFALLI, 2003, sec. 1.3.8)

9.2. Règles admissibles et variantes. On peut ajouter de nouvelles règles de déduction au système de règles de la figure 9.1 sans changer l'ensemble des formules prouvables.

9.2.1. Affaiblissement. Un premier exemple de règle admissible – que l'on trouvera souvent incluse dans les présentations des règles de la déduction naturelle – est la règle dite d'*affaiblissement*, pour tout ensemble Δ de formules :

$$\frac{\Gamma \vdash \varphi}{\Gamma, \Delta \vdash \varphi} \text{ (W)}$$

On peut en effet montrer par induction sur la hauteur des dérivation que, si $\Gamma \vdash_{\text{NK}_0} \varphi$, alors $\Gamma, \Delta \vdash_{\text{NK}_0} \varphi$, en ajoutant l'ensemble de formules Δ systématiquement à toutes les parties gauches des séquents. On fait pour cela une (laborieuse) distinction de cas selon la

■ (GOUBAULT-LARRECQ et MACKIE, 1997, lem. 2.20).

dernière règle employée ; on ne va traiter ici que quelques cas pour se convaincre qu'une telle démonstration est possible.

Pour la règle (ax) : Alors $\Gamma = \Gamma', \varphi$ et par (ax) on a aussi $\Gamma', \Delta, \varphi \vdash_{\text{NK}_0} \varphi$.

Pour la règle (\perp_c) : Alors il existe une dérivation de $\Gamma, \neg\varphi \vdash \perp$ de hauteur strictement plus petite, donc par hypothèse d'induction il en existe une de $\Gamma, \Delta, \neg\varphi \vdash \perp$ et par (\perp_c), on a aussi $\Gamma, \Delta \vdash_{\text{NK}_0} \varphi$.

Pour la règle (\vee_e) : Alors il existe trois dérivations montrant $\Gamma \vdash \delta \vee \gamma$, de $\Gamma, \delta \vdash \varphi$ et de $\Gamma, \gamma \vdash \varphi$, toutes trois de hauteur strictement inférieure, donc par hypothèse d'induction il existe aussi trois dérivations de $\Gamma, \Delta \vdash \delta \vee \gamma$, de $\Gamma, \Delta, \delta \vdash \varphi$ et de $\Gamma, \Delta, \gamma \vdash \varphi$, et par (\vee_e), on a aussi $\Gamma, \Delta \vdash_{\text{NK}_0} \varphi$.

9.2.2. *Substitutions propositionnelles.* Soit τ une substitution propositionnelle. On écrit $\Gamma\tau$ pour l'ensemble $\{\psi\tau \mid \psi \in \Gamma\}$ où l'on a appliqué la substitution τ systématiquement à toutes les formules de Γ . La règle ci-dessous de *substitution propositionnelle* est alors admissible :

$$\frac{\Gamma \vdash \varphi}{\Gamma\tau \vdash \varphi\tau} \text{ (S}_0\text{)}$$

On peut en effet montrer par induction sur la hauteur des dérivations que, si $\Gamma \vdash_{\text{NK}_0} \varphi$ et τ est une substitution propositionnelle, alors $\Gamma\tau \vdash_{\text{NK}_0} \varphi\tau$. On se convaincra aisément qu'il suffit d'appliquer τ systématiquement dans tous les séquents de la dérivation de $\Gamma \vdash \varphi$.

9.2.3. *Coupure.* La règle suivante est admissible :

$$\frac{\Gamma, \varphi \vdash \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi} \text{ (cut)}$$

En effet, elle peut se dériver simplement par

$$\frac{\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi} \text{ (}\rightarrow\text{i)}}{\Gamma \vdash \psi} \text{ (}\rightarrow\text{e)}$$

■ (DAVID, NOUR et RAFFALLI, 2003, prop. 1.3.18); voir le commentaire à côté de la démonstration du théorème 9.11 de complétude propositionnelle de la déduction naturelle page 134.

9.2.4. *Double négation.* On aurait pu remplacer dans le système de la figure 9.1 la règle (\perp_c) d'absurdité classique par la règle de *double négation* ci-dessous :

$$\frac{}{\Gamma, \neg\neg\varphi \vdash \varphi} \text{ (}\neg\neg\text{)}$$

En effet, cette règle est admissible car l'exemple 9.4 montre que $\neg\neg P \vdash_{\text{NK}_0} P$, puis (S_0) permet de déduire $\neg\neg\varphi \vdash_{\text{NK}_0} \varphi$ en appliquant la substitution propositionnelle $[\varphi/P]$, et la règle d'affaiblissement (W) permet de conclure.

Inversement, dans le système où l'on a remplacé (\perp_c) par ($\neg\neg$), on peut vérifier que (W) est encore admissible, et on a alors

$$\frac{\frac{\frac{}{\Gamma, \neg\neg\varphi \vdash \varphi} \text{ (}\neg\neg\text{)}}{\Gamma \vdash \neg\neg\varphi \rightarrow \varphi} \text{ (}\rightarrow\text{i)}}{\Gamma \vdash \varphi} \text{ (}\rightarrow\text{e)}$$

■ (DUPARC, 2015, ex. 134)

■ (DUPARC, 2015, ex. 131),
(DAVID, NOUR et RAFFALLI, 2003,
prop. 1.3.25)

9.2.5. *Absurdité intuitionniste.* La règle ci-dessous d'*absurdité intuitionniste* est une version plus faible de la règle (\perp_c) d'absurdité classique et est admissible :

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash \varphi} (\perp_i)$$

En effet, elle se dérive comme suit.

$$\frac{\frac{\Gamma \vdash \perp}{\Gamma, \neg\varphi \vdash \perp} (w)}{\Gamma \vdash \varphi} (\perp_c)$$

■ (DUPARC, 2015, ex. 132),
(DAVID, NOUR et RAFFALLI, 2003,
prop. 1.3.27)

9.2.6. *Tiers exclu.* La règle ci-dessous du *tiers exclu* découle elle aussi de la règle (\perp_c).

$$\frac{\Gamma, \varphi \vdash \psi \quad \Gamma, \neg\varphi \vdash \psi}{\Gamma \vdash \psi} (\text{te})$$

Elle peut en effet se dériver comme suit.

$$\frac{\frac{\frac{\frac{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \neg(\varphi \vee \neg\varphi)}{\Gamma, \neg(\varphi \vee \neg\varphi), \varphi \vdash \neg(\varphi \vee \neg\varphi)} (\text{ax})}{\Gamma, \neg(\varphi \vee \neg\varphi), \varphi \vdash \neg(\varphi \vee \neg\varphi)} (\text{ax})}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \neg(\varphi \vee \neg\varphi)} (\text{ax})}{\frac{\frac{\frac{\frac{\Gamma, \neg(\varphi \vee \neg\varphi), \varphi \vdash \perp}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \neg\varphi} (\neg_i)}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \neg\varphi} (\text{vi}_d)}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \varphi \vee \neg\varphi} (\neg_e)}{\Gamma \vdash \varphi \vee \neg\varphi} (\perp_c)}{\Gamma \vdash \psi} (\text{ve}) \quad \Gamma, \varphi \vdash \psi \quad \Gamma, \neg\varphi \vdash \psi$$

Inversement, dans un système sans (\perp_c) mais doté à la fois de (\perp_i) et de (te), la règle d'absurdité classique peut être dérivée.

$$\frac{\frac{\Gamma, \neg\varphi \vdash \perp}{\Gamma, \neg\varphi \vdash \varphi} (\text{ax})}{\Gamma \vdash \varphi} (\perp_i) \quad \frac{\Gamma, \neg\varphi \vdash \perp}{\Gamma, \neg\varphi \vdash \varphi} (\text{te})$$

9.2.7. *Contraposition.* La règle de *contraposition* ci-dessous est admissible, comme le montre la dérivation qui la suit.

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma, \neg\psi \vdash \neg\varphi} (\text{contra})$$

$$\frac{\frac{\frac{\Gamma, \varphi \vdash \psi}{\Gamma, \neg\psi, \varphi \vdash \neg\psi} (\text{ax})}{\Gamma, \neg\psi, \varphi \vdash \perp} (\neg_i)}{\Gamma, \neg\psi \vdash \neg\varphi} (\neg_e) \quad \frac{\Gamma, \varphi \vdash \psi}{\Gamma, \neg\psi, \varphi \vdash \psi} (w)$$

■ (DUPARC, 2015, ex. 135),
(DAVID, NOUR et RAFFALLI, 2003,
prop. 1.3.28)

La direction opposée (cc) de la règle de contraposition est elle aussi admissible :

$$\frac{\Gamma, \neg\varphi \vdash \neg\psi}{\Gamma, \psi \vdash \varphi} (\text{cc})$$

Sa dérivation est quasiment identique mais nécessite cette fois d'utiliser l'absurdité classique :

$$\frac{\frac{\Gamma, \psi, \neg\varphi \vdash \psi \quad (\text{ax}) \quad \frac{\Gamma, \neg\varphi \vdash \neg\psi \quad (\text{W})}{\Gamma, \psi, \neg\varphi \vdash \neg\psi} \quad (\neg\text{e})}{\Gamma, \psi, \neg\varphi \vdash \perp} \quad (\perp\text{c})}{\Gamma, \psi \vdash \varphi} \quad (\perp\text{c})$$

Inversement, dans un système sans la règle $(\perp\text{c})$, l'absurdité classique est admissible pour peu que l'on se dote de la règle (cc) :

$$\frac{\frac{\frac{\frac{\perp, \neg\perp \vdash \perp \quad (\text{ax})}{\perp, \neg\perp \vdash \neg\perp} \quad (\neg\text{e})}{\perp, \neg\perp \vdash \perp} \quad (\neg\text{i})}{\perp \vdash \neg\neg\perp} \quad (\text{W})}{\Gamma, \neg\varphi, \perp \vdash \neg\neg\perp} \quad (\text{W})}{\Gamma, \neg\varphi \vdash \neg\neg\perp} \quad (\text{cc})}{\Gamma, \neg\perp \vdash \varphi} \quad (\text{cut})}{\Gamma \vdash \varphi} \quad (\text{cut}) \quad \frac{\frac{\frac{\perp \vdash \perp \quad (\text{ax})}{\vdash \neg\perp} \quad (\neg\text{i})}{\Gamma \vdash \neg\perp} \quad (\text{W})}{\Gamma \vdash \varphi} \quad (\text{cut})$$

9.3. Correction et complétude. Un séquent $\Gamma \vdash \varphi$ est *valide* si φ est une conséquence logique de Γ , c'est-à-dire si $\Gamma \models \varphi$; dans le cas où $\Gamma = \emptyset$, cela correspond bien à la validité de la formule φ . Les théorèmes de correction et de complétude montrent que les séquents valides sont exactement les séquents prouvables en déduction naturelle classique propositionnelle.

La correction du système de déduction de la figure 9.1 montre que toute formule prouvable, c'est-à-dire telle que $\vdash_{\text{NK}_0} \varphi$, est bien valide.

Théorème 9.10 (correction propositionnelle de la déduction naturelle). *Soit Γ un ensemble fini de formules propositionnelles et φ une formule propositionnelle. Si $\Gamma \vdash_{\text{NK}_0} \varphi$, alors $\Gamma \models \varphi$.*

■ (DUPARC, 2015, th. 139 pp. 207–210), (JAUME et al., 2020, thm. 5.1)

Démonstration. On va montrer que toutes les règles de la figure 9.1 sont *correctes* au sens suivant : si tous les séquents prémisses sont valides, alors le séquent conclusion est lui aussi valide. On ne va pas traiter tous les cas, mais un échantillon :

Règle (ax) : Cette règle n'a pas de prémisses et on montre directement que sa conclusion est valide. Si $I \models \Gamma \cup \{\varphi\}$, alors en particulier $I \models \varphi$: on a bien $\Gamma \cup \{\varphi\} \models \varphi$.

Règle $(\perp\text{c})$: Supposons que la prémisse $\Gamma, \neg\varphi \vdash \perp$ soit valide, c'est-à-dire que $\Gamma \cup \{\neg\varphi\} \models \perp$. Par le lemme 6.2 de déduction, $\Gamma \cup \{\neg\varphi\} \models \perp$ si et seulement si $\Gamma \models \neg\varphi \rightarrow \perp$, où $(\neg\varphi \rightarrow \perp) \leftrightarrow \varphi$; cela montre que la conclusion de la règle $\Gamma \vdash \varphi$ est elle aussi valide.

Règle $(\neg\text{e})$: Supposons que les deux prémisses $\Gamma \vdash \neg\varphi$ et $\Gamma \vdash \varphi$ soient valides, c'est-à-dire que $\Gamma \models \neg\varphi$ et $\Gamma \models \varphi$. Supposons qu'il existe une interprétation I telle que $I \models \Gamma$. Alors $I \not\models \varphi$ et $I \models \varphi$, ce qui est absurde : une telle interprétation I n'existe pas, autrement dit Γ est insatisfiable : $\Gamma \models \perp$. Le séquent conclusion $\Gamma \vdash \perp$ est donc bien valide.

Règle $(\vee\text{e})$: Supposons que les trois prémisses $\Gamma \vdash \varphi \vee \psi$, $\Gamma, \varphi \vdash \delta$ et $\Gamma, \psi \vdash \delta$ soient toutes trois valides. Soit I une interprétation telle que $I \models \Gamma$. Comme $\Gamma \models \varphi \vee \psi$, on a alors $I \models \varphi$ ou $I \models \psi$:

- dans le premier cas, on a aussi $I \models \delta$ puisque $\Gamma \cup \{\varphi\} \models \delta$;
- dans le second cas, on a aussi $I \models \delta$ puisque $\Gamma \cup \{\psi\} \models \delta$.

On a donc $\Gamma \vdash \delta$ valide dans tous les cas.

Règle (\wedge i) : Supposons que les deux prémisses $\Gamma \vdash \varphi$ et $\Gamma \vdash \psi$ soient valides, c'est-à-dire que $\Gamma \models \varphi$ et $\Gamma \models \psi$. Soit I une interprétation telle que $I \models \Gamma$. Alors $I \models \varphi$ et $I \models \psi$, et donc $I \models \varphi \wedge \psi$. Cela montre que le séquent conclusion $\Gamma \vdash \varphi \wedge \psi$ est valide.

Règle (\rightarrow i) : Supposons que la prémisse $\Gamma, \varphi \vdash \psi$ est valide, c'est-à-dire que $\Gamma \cup \{\varphi\} \models \psi$. Par le lemme 6.2 de déduction, $\Gamma \models \varphi \rightarrow \psi$ et le séquent conclusion $\Gamma \vdash \varphi \rightarrow \psi$ est donc bien valide.

On démontre ensuite le théorème par induction sur un arbre de dérivation du séquent prouvable $\Gamma \vdash \varphi$. Cette dérivation est de la forme suivante pour un certain $0 \leq k \leq 3$ et une règle (R) de la figure 9.1 :

$$\frac{\begin{array}{ccc} \pi_1 & & \pi_k \\ \vdots & & \vdots \\ \Gamma_1 \vdash \varphi_1 & \cdots & \Gamma_k \vdash \varphi_k \end{array}}{\Gamma \vdash \varphi} \text{ (R)}$$

Par hypothèse d'induction, chacun des séquents $\Gamma_1 \vdash \varphi_1, \dots, \Gamma_k \vdash \varphi_k$ est valide, et comme la règle (R) est correcte, $\Gamma \vdash \varphi$ est lui aussi valide. \square

Inversement, pour toute formule propositionnelle valide φ , il existe une dérivation montrant $\vdash_{\text{NK}_0} \varphi$ dans le système de la figure 9.1. Plus généralement, on a le résultat suivant.

■ Voir (GOUBAULT-LARRECQ et MACKIE, 1997, th. 2.26) pour une démonstration à l'aide d'arbres sémantiques et (DUPARC, 2015, th. 139 pp. 210–215) pour une approche plus traditionnelle.

Théorème 9.11 (complétude propositionnelle de la déduction naturelle). *Soit Γ un ensemble fini de formules propositionnelles et φ une formule propositionnelle. Si $\Gamma \models \varphi$, alors $\Gamma \vdash_{\text{NK}_0} \varphi$.*

On ne va pas fournir de démonstration directe du théorème 9.11 ici, mais on l'obtiendra comme corollaire des résultats de la section 11.4 sur l'automatisation de la recherche de preuves en logique classique propositionnelle.

9.4. Assistants de preuve. Une excellente manière de s'entraîner à faire des démonstrations en déduction naturelle est d'employer un *assistant de preuve*, qui va vérifier de manière interactive que chaque application de règle respecte bien sa définition, indiquer quels séquents restent encore à démontrer, rappeler quelles hypothèses ont été faites, ... De nombreux assistants de preuve existent, dont certains à vocation spécifiquement pédagogique comme par exemple *edukera*⁵, et d'autres employés pour des projets de taille industrielle comme par exemple *Coq*⁶, qui est le système que nous allons explorer dans ces notes.

On ne va donner ici qu'un tout petit aperçu de ce qui est possible en *Coq*; on pourra par exemple consulter des tutoriels⁷ pour se familiariser avec le système. Un *contexte* Γ va maintenant être un ensemble fini de paires $h : \varphi$ où φ est une formule et h est un identifiant. Les séquents sont des paires $\Gamma \vdash \varphi$ où Γ est un contexte et φ une formule.

En *Coq*, on utilise des *tactiques* pour progresser dans la recherche de preuve; essentiellement, une tactique peut être vue comme une règle de déduction : si l'objectif courant instancie la conclusion de la règle, on peut l'appliquer et les prémisses de la règles deviennent les

5. <https://app.edukera.com>

6. <https://coq.inria.fr>

7. <https://coq.inria.fr/tutorial/1-basic-predicate-calculus>

nouveaux objectifs de preuve. La figure 9.2 présente quelques-unes des tactiques disponibles dans Coq.

$$\begin{array}{c}
\frac{}{\Gamma, h : \varphi \vdash \varphi} \text{ (assumption)} \\
\frac{\Gamma \vdash \perp}{\Gamma \vdash \varphi} \text{ (exfalso)} \\
\frac{\Gamma, h : \varphi \vdash \perp}{\Gamma \vdash \neg \varphi} \text{ (intro h)} \\
\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \text{ (left)} \quad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi} \text{ (right)} \\
\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi} \text{ (split)} \\
\frac{\Gamma, h : \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi} \text{ (intro h)} \\
\frac{\Gamma \vdash \psi}{\Gamma, h : \varphi \vdash \psi} \text{ (clear h)} \\
\frac{}{\Gamma \vdash \varphi \vee \neg \varphi} \text{ (exact(classic } \varphi)) \\
\frac{\Gamma \vdash \varphi \quad \Gamma, h : \varphi \vdash \psi}{\Gamma \vdash \psi} \text{ (assert (h : } \varphi)) \\
\frac{\Gamma \vdash \varphi}{\Gamma, h : \neg \varphi \vdash \psi} \text{ (destruct h)} \\
\frac{\Gamma, l : \varphi \vdash \delta \quad \Gamma, r : \psi \vdash \delta}{\Gamma, h : \varphi \vee \psi \vdash \delta} \text{ (destruct h as [l|r])} \\
\frac{\Gamma, l : \varphi, r : \psi \vdash \delta}{\Gamma, h : \varphi \wedge \psi \vdash \delta} \text{ (destruct h as (l, r))} \\
\frac{\Gamma, h : \varphi \rightarrow \psi \vdash \varphi}{\Gamma, h : \varphi \rightarrow \psi \vdash \psi} \text{ (apply h)}
\end{array}$$

FIGURE 9.2. Quelques-unes des tactiques propositionnelles disponibles en Coq.

On retrouve dans la colonne de gauche de la figure 9.2 des règles familières de la déduction naturelle propositionnelle : aux identifiants près, de haut en bas, (assumption) n'est autre que (ax), (exfalso) est (\perp_i), (intro h) est (\neg_i), (left) et (right) sont (\vee_g) et (\vee_d), (split) est (\wedge_i), (intro h) est (\rightarrow_i), et (clear h) est (W). Les règles de la colonne de droite sont quant à elles différentes de celles de la déduction naturelle, à l'exception de (assert (h : φ)), qui n'est autre que (cut).

9.4.1. \square *Exemples de preuves en Coq.* Voyons quelques exemples de preuves en Coq. Il est recommandé de les tester dans un environnement interactif comme proofgeneral⁸ sous Emacs ou coqide⁹. On commence par importer le module de raisonnement pour la logique classique et par déclarer les noms de propositions P , Q , R que nous allons utiliser dans nos exemples.

```

propositional.v
Require Import Classical.
Variables P Q R : Prop.

```

Exemple 9.1. Voici maintenant une preuve en Coq de la prouvabilité de la formule $(P \wedge (P \rightarrow Q)) \rightarrow Q$, ce qui correspond au séquent de l'exemple 9.1.

```

propositional.v
(* Exemple 9.1. modus ponens *)
Goal P /\ (P -> Q) -> Q.
intro h.
destruct h as (p, piq).

```

8. <https://proofgeneral.github.io/>

9. <https://coq.inria.fr/refman/practical-tools/coqide.html>

```

apply piq.
assumption.
Qed.

```

Cette séquence de tactiques correspond en fait à l'exploration d'un arbre de dérivation depuis l'objectif initial $\vdash (P \wedge (P \rightarrow Q)) \rightarrow Q$, par applications successives des règles de la figure 9.2. En mode interactif, on voit s'afficher le contenu du contexte Γ et les différents séquents qu'il reste à prouver. Cette séquence de tactiques correspond à la dérivation suivante avec les règles de la figure 9.2.

$$\frac{\frac{\frac{\frac{}{p : P, \text{piq} : P \rightarrow Q \vdash P}}{\text{p} : P, \text{piq} : P \rightarrow Q \vdash P}}{\text{p} : P, \text{piq} : P \rightarrow Q \vdash Q}}{\text{h} : P \wedge (P \rightarrow Q) \vdash Q}}{\vdash P \wedge (P \rightarrow Q) \rightarrow Q}$$

(assumption)
(apply piq)
(destruct h as (p,piq))
(intro h)

Exemple 9.2. Voici une preuve de la formule $((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R)$, qui correspond au séquent de l'exemple 9.2.

```

propositional.v
(* Exemple 9.2. barbara *)
Goal (P -> Q) /\ (Q -> R) -> (P -> R).
intro h.
destruct h as (piq, qir).
intro p.
apply qir.
apply piq.
assumption.
Qed.

```

La dérivation correspondante utilisant les règles de la figure 9.2 est la suivante.

$$\frac{\frac{\frac{\frac{\frac{}{\text{piq} : P \rightarrow Q, \text{qir} : Q \rightarrow R, \text{p} : P \vdash Q}}{\text{piq} : P \rightarrow Q, \text{qir} : Q \rightarrow R, \text{p} : P \vdash Q}}{\text{piq} : P \rightarrow Q, \text{qir} : Q \rightarrow R, \text{p} : P \vdash R}}{\text{piq} : P \rightarrow Q, \text{qir} : Q \rightarrow R \vdash P \rightarrow R}}{\text{h} : (P \rightarrow Q) \wedge (Q \rightarrow R) \vdash P \rightarrow R}}{\vdash (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow P \rightarrow R}$$

(assumption)
(apply piq)
(apply qir)
(intro p)
(destruct h as (piq,qir))
(intro h)

Exemple 9.3. En guise de troisième exemple, voici une preuve de la formule de l'exemple 9.3.

```

propositional.v
(* Exemple 9.3. *)
Goal (P -> Q) -> ~ (P /\ ~ Q).
intro piq.
intro penq.
destruct penq as (p, nq).
destruct nq.
apply piq.
assumption.
Qed.

```

La dérivation qui est construite ici est la suivante.

$$\begin{array}{c}
\frac{}{\text{piq} : P \rightarrow Q, p : P \vdash P} \text{(assumption)} \\
\frac{}{\text{piq} : P \rightarrow Q, p : P \vdash Q} \text{(apply piqu)} \\
\frac{}{\text{piq} : P \rightarrow Q, p : P, \text{ng} : \neg Q \vdash \perp} \text{(destruct ng)} \\
\frac{}{\text{piq} : P \rightarrow Q, \text{penq} : P \wedge \neg Q \vdash \perp} \text{(destruct penq as (p,ng))} \\
\frac{}{\text{piq} : P \rightarrow Q \vdash \neg(P \wedge \neg Q)} \text{(intro penq)} \\
\frac{}{\vdash (P \rightarrow Q) \rightarrow \neg(P \wedge \neg Q)} \text{(intro piqu)}
\end{array}$$

Loi de PEIRCE. Nous n'avons jusqu'ici pas eu besoin d'employer la règle ($\text{exact}(\text{classic } \varphi)$). La loi de PEIRCE est un exemple de formule valide qui va nécessiter son emploi.

propositional.v

```

(* Exemple 9.5. loi de Peirce *)
Goal ((P -> Q) -> P) -> P.
intro piquip.
assert (ponp : P \ / ~P).
exact (classic P).
destruct ponp as [p|np].
assumption.
apply piquip.
intro p.
destruct np.
assumption.
Qed.

```

La dérivation correspondante avec les règles de la figure 9.2 est la suivante.

$$\frac{}{\text{exact}(\text{classic } P) \quad \text{piqip} : (P \rightarrow Q) \rightarrow P \vdash P \vee \neg P} \text{(assumption)} \quad \frac{}{\text{piqip} : (P \rightarrow Q) \rightarrow P, p : P \vdash P} \text{(assumption)} \quad \frac{}{\text{piqip} : (P \rightarrow Q) \rightarrow P, np : \neg P, p : P \vdash Q} \text{(destruct np)} \quad \frac{}{\text{piqip} : (P \rightarrow Q) \rightarrow P, np : \neg P \vdash P \rightarrow Q} \text{(intro p)} \quad \frac{}{\text{piqip} : (P \rightarrow Q) \rightarrow P, np : \neg P \vdash P} \text{(apply piquip)} \quad \frac{}{\text{piqip} : (P \rightarrow Q) \rightarrow P, \text{ponp} : P \vee \neg P \vdash P} \text{(destruct ponp as [p|np])} \quad \frac{}{\text{piqip} : (P \rightarrow Q) \rightarrow P \vdash P} \text{(intro piquip)} \quad \frac{}{\vdash ((P \rightarrow Q) \rightarrow P) \rightarrow P} \text{(assert (ponp : P \vee \neg P))}$$

9.4.2. ** Admissibilité des tactiques de Coq.* Notre objectif dans cette sous-section va être de montrer que les règles de la figure 9.2 sont toutes admissibles en déduction naturelle. De par le théorème 9.10 de correction propositionnelle de la déduction naturelle, toutes les formules dérivables à l'aide des règles de la figure 9.2 sont donc bien valides.

Comme mentionné juste en-dessous de la figure 9.2, la majorité des règles sont des règles déjà présentes dans la figure 9.1 – à savoir (assumption), (intro h), (left), (right), (split) et (intro h) – ou bien ont déjà été montrées admissibles dans la section 9.2 – c'est le cas de (exfalse), (clear h) et ($\text{assert } (h : \varphi)$).

Il reste à montrer que les cinq règles ($\text{exact}(\text{classic } \varphi)$), (destruct h), ($\text{destruct h as [l|r]}$), ($\text{destruct h as (l, r)}$) et (apply h) sont admissibles. Parmi celles-ci, on verra que les deux règles ($\text{destruct h as [l|r]}$) et ($\text{destruct h as (l, r)}$) correspondent aux règles d'introduction à gauche (\vee) et (\wedge) qui seront montrées admissibles dans la proposition 11.15 en section 11.4.1.

Admissibilité de ($\text{exact}(\text{classic } \varphi)$). Commençons par la règle de tiers exclu ($\text{exact}(\text{classic } \varphi)$). Cette règle est réminiscente de la règle de tiers exclu, et peut être dérivée de manière similaire dans le système de la figure 9.1.

$$\begin{array}{c}
\frac{\Gamma, \neg(\varphi \vee \neg\varphi), \varphi \vdash \neg(\varphi \vee \neg\varphi) \quad \frac{\Gamma, \neg(\varphi \vee \neg\varphi), \varphi \vdash \varphi \quad \Gamma, \neg(\varphi \vee \neg\varphi) \vdash \varphi \vee \neg\varphi}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \varphi \vee \neg\varphi} \text{(ax)} \quad \frac{\Gamma, \neg(\varphi \vee \neg\varphi), \varphi \vdash \varphi \quad \Gamma, \neg(\varphi \vee \neg\varphi) \vdash \varphi \vee \neg\varphi}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \varphi \vee \neg\varphi} \text{(v_ig)}}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \neg(\varphi \vee \neg\varphi)} \text{(ax)} \\
\frac{\Gamma, \neg(\varphi \vee \neg\varphi), \varphi \vdash \perp \quad \Gamma, \neg(\varphi \vee \neg\varphi) \vdash \neg\varphi}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \varphi \vee \neg\varphi} \text{(v_id)} \quad \frac{\Gamma, \neg(\varphi \vee \neg\varphi), \varphi \vdash \perp}{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \neg\varphi} \text{(¬-i)} \\
\frac{\Gamma, \neg(\varphi \vee \neg\varphi) \vdash \perp}{\Gamma \vdash \varphi \vee \neg\varphi} \text{(¬-c)}
\end{array}$$

Admissibilité de (destruct h). Considérons maintenant la règle (destruct h). Voici une dérivation en déduction naturelle qui la montre admissible.

$$\frac{\Gamma, \neg\varphi \vdash \neg\varphi \quad \Gamma \vdash \varphi}{\Gamma, \neg\varphi \vdash \varphi} \text{(ax)} \quad \frac{\Gamma, \neg\varphi \vdash \perp}{\Gamma, \neg\varphi \vdash \psi} \text{(¬-i)}$$

Admissibilité de (apply h). Pour finir, voici une dérivation en déduction naturelle qui montre que (apply h) est admissible.

$$\frac{\Gamma, \varphi \rightarrow \psi, \varphi \vdash \varphi \rightarrow \psi \quad \Gamma, \varphi \rightarrow \psi, \varphi \vdash \varphi}{\Gamma, \varphi \rightarrow \psi, \varphi \vdash \psi} \text{(ax)} \quad \frac{\Gamma, \varphi \rightarrow \psi, \varphi \vdash \varphi \quad \Gamma, \varphi \rightarrow \psi \vdash \varphi}{\Gamma, \varphi \rightarrow \psi \vdash \varphi} \text{(¬-e)}$$

9.4.3. * \square Règles de la déduction naturelle en Coq. Nous venons de vérifier en section 9.4.2 que les règles de la figure 9.2 étaient bien admissibles en déduction naturelle. Nous allons voir ici qu'inversement, les règles de la déduction naturelle données en figure 9.1 peuvent être exprimées à l'aide des règles de la figure 9.2. Par le théorème 9.11 de complétude propositionnelle de la déduction naturelle, cela montrera que toutes les formules propositionnelles valides ont bien une preuve en Coq utilisant les règles dérivables à l'aide des règles de la figure 9.2. Mieux encore : on peut écrire de nouvelles tactiques pour Coq qui vont implémenter *exactement* les règles de la figure 9.1.

Règles d'introduction. On peut commencer par ajouter des noms pour les règles d'introduction de la figure 9.1 ainsi que la règle d'absurdité intuitionniste, qui ont toutes une correspondance directe avec des règles de la figure 9.2. On utilise pour cela « Tactic Notation »¹⁰ qui permet de définir de nouveaux noms de tactiques.

```

naturaldeduction.v
Require Import Classical.

Tactic Notation "ax" := assumption.
Tactic Notation "neg_i" := intro.
Tactic Notation "neg_i" ident(x) := intro x.
Tactic Notation "ou_ig" := left.
Tactic Notation "ou_id" := right.
Tactic Notation "et_i" := split.
Tactic Notation "impl_i" := intro.
Tactic Notation "impl_i" ident(x) := intro x.
Tactic Notation "absurdite_i" := exfalso.

```

10. <https://coq.inria.fr/refman/user-extensions/syntax-extensions.html>

Pour les autres règles de la figure 9.1, on va définir de nouvelles tactiques comme des compositions de tactiques existantes à l'aide de « Ltac »¹¹.

Règle (\neg e). Ainsi, pour la règle d'élimination de \neg , on définit la tactique suivante.

```
naturaldeduction.v
(* Règle ( $\neg$ e) *)
Ltac neg_e P := match goal with
| - False => let H := fresh in
              assert (H:~ P);
              [| destruct H]
end.
```

Cette tactique vérifie que le conséquent du séquent conclusion est bien \perp , et dans ce cas emploie la règle (`assert (h : φ)`) puis (`destruct h`) sur sa prémisse droite. La dérivation correspondante avec les règles de la figure 9.2 est la suivante.

$$\frac{\Gamma \vdash \neg\varphi \quad \frac{\Gamma \vdash \varphi}{\Gamma, H : \neg\varphi \vdash \perp} \text{(destruct H)}}{\Gamma \vdash \perp} \text{(assert (H:}\neg\varphi\text{))}$$

Règle (\vee e). Voici comment définir une tactique qui implémente la règle d'élimination de \vee .

```
naturaldeduction.v
(* Règle ( $\vee$ e) *)
Ltac ou_e_tactic A l r :=
  let H := fresh in
  assert (H:A);
  [| destruct H as [l|r]].
Tactic Notation "ou_e" constr(A) :=
  let H0 := fresh in
  let H1 := fresh in
  ou_e_tactic A H0 H1.
Tactic Notation "ou_e" constr(A) "as" "[" ident(y) "|" ident(z) "]" :=
  ou_e_tactic A y z.
```

La première partie de l'implémentation définit une tactique `ou_e_tactic` en commençant par employer (`assert (h : φ)`) puis (`destruct h as [l|r]`) sur sa prémisse droite. La seconde partie de l'implémentation laisse le choix à l'utilisateur entre utiliser des identifiants générés automatiquement ou fournis explicitement en argument. La dérivation avec les règles de la figure 9.2 correspondant à cette implémentation est la suivante.

$$\frac{\Gamma \vdash \varphi \vee \psi \quad \frac{\Gamma, l : \varphi \vdash \delta \quad \Gamma, r : \psi \vdash \delta}{\Gamma, H : \varphi \vee \psi \vdash \delta} \text{(destruct H as [l|r])}}{\Gamma \vdash \delta} \text{(assert (H:}\varphi \vee \psi\text{))}$$

Règle (\wedge e_g). L'implémentation ci-dessous de la règle d'élimination de \wedge trouve la formule objectif B grâce à `match goal with`.

```
naturaldeduction.v
(* Règle ( $\wedge$ eg) *)
Ltac et_eg A :=
  let H := fresh in
```

11. <https://coq.inria.fr/refman/proof-engine/ltac.html>

```

let H0 := fresh in
let H1 := fresh in
match goal with
|- ?B => assert (H:B /\ A);
      [| destruct H as (H0,H1); assumption ]
end.

```

La dérivation correspondante à l'aide des règles de la figure 9.2 est la suivante.

$$\frac{\Gamma \vdash \varphi \wedge \psi \quad \frac{\Gamma, H_0 : \varphi, H_1 : \psi \vdash \varphi \quad (\text{destruct } H \text{ as } (H_0, H_1))}{\Gamma, H : \varphi \wedge \psi \vdash \varphi} \text{(assumption)}}{\Gamma \vdash \varphi} \text{(assert (H : } \varphi \wedge \psi))$$

Règle (\rightarrow e). Voici une implémentation de la règle d'élimination de \rightarrow sous la forme d'une tactique Coq.

```

naturaldeduction.v
(* Règle ( $\rightarrow$ e) *)
Ltac impl_e A :=
let H := fresh in
match goal with
|- ?B => assert (H: A  $\rightarrow$  B);
      [| apply H; clear H ]
end.

```

La dérivation correspondante est la suivante.

$$\frac{\Gamma \vdash \varphi \rightarrow \psi \quad \frac{\Gamma \vdash \varphi \quad \Gamma, H : \varphi \rightarrow \psi \vdash \varphi}{\Gamma, H : \varphi \rightarrow \psi \vdash \psi} \text{(clear H)}}{\Gamma \vdash \psi} \text{(apply H)} \text{(assert (H : } \varphi \rightarrow \psi))$$

Règle (te). Avant de donner l'implémentation de la règle d'absurdité classique, nous commençons par implémenter la règle du tiers exclu comme suit.

```

naturaldeduction.v
(* Règle (te) *)
Ltac te_tactic A l r :=
let H := fresh in
assert (H: A \ / ~ A);
  [ exact (classic A) | destruct H as [l|r] ].
Tactic Notation "te" constr(A) :=
let H0 := fresh in
let H1 := fresh in
  te_tactic A H0 H1.
Tactic Notation "te" constr(A) "as" "[" ident(y) "|" ident(z) "]" :=
  te_tactic A y z.

```

La dérivation correspondante est la suivante.

$$\frac{\text{(exact (classic } \varphi)) \quad \frac{\Gamma \vdash \varphi \vee \neg \varphi \quad \Gamma, l : \varphi \vdash \psi \quad \Gamma, r : \neg \varphi \vdash \psi}{\Gamma, H : \varphi \vee \neg \varphi \vdash \psi} \text{(destruct H as [l|r])}}{\Gamma \vdash \psi} \text{(assert (H : } \varphi \vee \neg \varphi))$$

Règle (\perp_c). Finissons avec la règle d'absurdité classique, qu'on implémente à l'aide de (te) comme suit.

```
naturaldeduction.v
(* Règle (Bottom_c) *)
Lemma absc : forall A : Prop, (~ A -> False) -> A.
intro P.
intro npib.
te P as [p|np].
assumption.
exfalse.
apply npib.
assumption.
Qed.
Ltac absurdite_c h := apply absc; intro h.
```

On commence ici par montrer la validité de la formule $(\neg\varphi \rightarrow \perp) \rightarrow \varphi$, à partir de laquelle une simple utilisation de (intro h) donnera la tactique désirée. La dérivation de notre formule se montre comme suit.

$$\begin{array}{c}
 \frac{}{\text{npib} : \neg\varphi \rightarrow \perp, \text{np} : \neg\varphi \vdash \neg\varphi} \text{(assumption)} \\
 \frac{}{\text{npib} : \neg\varphi \rightarrow \perp, \text{np} : \neg\varphi \vdash \perp} \text{(apply npib)} \\
 \frac{}{\text{npib} : \neg\varphi \rightarrow \perp, \text{np} : \neg\varphi \vdash \varphi} \text{(exfalse)} \\
 \frac{}{\text{npib} : \neg\varphi \rightarrow \perp, \text{p} : \varphi \vdash \varphi} \text{(te } \varphi \text{ as [p|np])} \\
 \frac{}{\text{npib} : \neg\varphi \rightarrow \perp \vdash \varphi} \text{(intro npib)} \\
 \vdash (\neg\varphi \rightarrow \perp) \rightarrow \varphi
 \end{array}$$

Exemple d'utilisation. Toutes les nouvelles tactiques que nous avons implémentées permettent de vérifier directement des dérivations en déduction naturelle dans COQ. Ainsi, la dérivation de l'exemple 9.9 est prouvée en COQ comme suit.

```
naturaldeduction.v
(* Exemple 9.9. décurryfication *)
Goal (P -> (Q -> R)) -> ((P /\ Q) -> R).
impl_i.
impl_i.
impl_e Q.
impl_e P.
ax.
et_eg Q.
ax.
et_ed P.
ax.
Qed.
```


10. DÉDUCTION NATURELLE CLASSIQUE DU PREMIER ORDRE

Résumé. Le système de règles de la déduction naturelle classique propositionnelle peut être étendu à la logique du premier ordre à l'aide de règles d'introduction et d'élimination des quantificateurs. Cela permet de raisonner sur des séquents $\Gamma \vdash \varphi$ où les formules de Γ et la formule φ sont des formules de la logique du premier ordre. Une attention particulière doit être portée au traitement des variables libres et liées dans ces règles (voir section 10.1). Un séquent $\Gamma \vdash \varphi$ pour lequel il existe une dérivation dans le système de déduction est dit *prouvable*, ce qui est noté « $\Gamma \vdash_{\text{NK}} \varphi$ ».

Comme dans le cas propositionnel, un séquent $\Gamma \vdash \varphi$ est *valide* si $\Gamma \models \varphi$. Par les théorèmes 10.9 et 10.10 de correction et de complétude, un séquent est valide si et seulement s'il est prouvable.

Comme dans le cas de la déduction naturelle classique propositionnelle en section 9, nous allons travailler avec une présentation sous la forme d'un système de règles de déduction qui travaillent sur des *séquents* $\Gamma \vdash \varphi$, où Γ est un ensemble de formules et φ une formule, et en utilisant une syntaxe étendue

$$t ::= x \mid f(t, \dots, t)$$

$$\varphi ::= R(t, \dots, t) \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \rightarrow \varphi \mid \exists x.\varphi \mid \forall x.\varphi$$

où $x \in X$, $f \in \mathcal{F}$ et $R \in \mathcal{P}$ sont des symboles de variable, de fonction, et de relation.

Les règles de la déduction naturelle du premier ordre sont alors celles de la figure 9.1 complétées par une règle d'introduction et une règle d'élimination pour chaque symbole de quantification, et données dans la figure 10.1.

■ (DUPARC, 2015, sec. III.13.2),
(DAVID, NOUR et RAFFALLI, 2003,
sec. 1.3), (GOUBAULT-LARRECQ et
MACKIE, 1997, fig. 6.1)

$$\frac{\Gamma \vdash \varphi[t/x]}{\Gamma \vdash \exists x.\varphi} (\exists i) \quad \text{où } t \in T(\mathcal{F}, X)$$

$$\frac{\Gamma \vdash \exists x.\varphi \quad \Gamma, \varphi[y/x] \vdash \psi}{\Gamma \vdash \psi} (\exists e) \quad \text{où } y \notin \text{Libres}(\Gamma, \exists x.\varphi, \psi)$$

$$\frac{\Gamma \vdash \varphi[y/x]}{\Gamma \vdash \forall x.\varphi} (\forall i) \quad \text{où } y \notin \text{Libres}(\Gamma, \forall x.\varphi)$$

$$\frac{\Gamma \vdash \forall x.\varphi}{\Gamma \vdash \varphi[t/x]} (\forall e) \quad \text{où } t \in T(\mathcal{F}, X)$$

FIGURE 10.1. Règles de quantification en déduction naturelle classique du premier ordre ; les substitutions doivent être applicables.

Un séquent $\Gamma \vdash \varphi$ pour lequel il existe une dérivation dans le système de déduction des figures 9.1 et 10.1 est dit *prouvable* et on note alors « $\Gamma \vdash_{\text{NK}} \varphi$ ».

Exemple 10.1. Comme dans le cas propositionnel, les dérivations en déduction naturelle reflètent assez fidèlement les étapes d'une démonstration mathématique. Nous allons illustrer cela pour la formule $\forall x.\varphi \rightarrow \exists x.\varphi$. Celle-ci peut être dérivée comme ci-dessous, où la règle (cut) de coupure reste bien admissible en déduction naturelle du premier ordre, et où les substitutions $[t/x]$ des applications des règles ($\forall e$) et ($\exists i$) utilisent $t \stackrel{\text{def}}{=} x$ et sont donc l'identité.

$$\frac{\frac{\frac{}{\forall x.\varphi, \varphi \vdash \varphi} \text{(ax)}}{\forall x.\varphi, \varphi \vdash \exists x.\varphi} \text{(}\exists i\text{)}}{\frac{\frac{\frac{}{\forall x.\varphi \vdash \forall x.\varphi} \text{(ax)}}{\forall x.\varphi \vdash \varphi} \text{(}\forall e\text{)}}{\vdash \forall x.\varphi \rightarrow \exists x.\varphi} \text{(cut)}} \text{(}\rightarrow i\text{)}$$

Le raisonnement sous-jacent à cette dérivation est le suivant. Pour montrer que $\forall x.\varphi \rightarrow \exists x.\varphi$ est valide, par le lemme 6.2 de déduction, il suffit de montrer que, pour toute interprétation I et toute valuation ρ , si $I, \rho \models \forall x.\varphi$ alors $I, \rho \models \exists x.\varphi$ ($\rightarrow i$). On montre pour cela (cut)

- d'une part que $I, \rho \models \forall x.\varphi$ (ax) implique qu'en particulier, pour l'élément $e \stackrel{\text{def}}{=} \rho(x)$ issu du domaine $D_I \neq \emptyset$, $I, \rho \models \varphi$ ($\forall e$) et
- d'autre part que si $I, \rho \models \varphi$, alors cet élément $e = \rho(x)$ permet de déduire que $I, \rho \models \exists x.\varphi$ ($\exists i$).

Exemple 10.2 (Formule du buveur). Voyons un autre exemple dans le cas de la formule du buveur « il existe un individu tel que s'il boit alors tout le monde boit » qui avait été montrée valide dans l'exemple 2.8. Notre objectif est de montrer que la formule $\varphi \stackrel{\text{def}}{=} \exists x.(B(x) \rightarrow \forall y.B(y))$ est valide, c'est-à-dire que dans toute interprétation I et pour toute valuation ρ , on a $I, \rho \models \varphi$.

On commence par noter que la règle (te) du tiers exclu reste bien admissible en déduction naturelle au premier ordre. La première étape dans l'exemple 2.8 est alors d'utiliser le tiers exclu (te) en distinguant deux cas :

- soit $I, \rho \models \exists x.\neg B(x)$, autrement dit il existe une personne qui ne boit pas : $B^I \subsetneq D_I$;
- soit $I, \rho \models \neg(\exists x.\neg B(x))$, autrement dit il n'existe pas de personne qui ne boit pas : $B^I = D_I$.

Cela correspond à la première étape de recherche de preuve ci-dessous.

$$\frac{\begin{array}{c} \vdots \\ \exists x.\neg B(x) \vdash \varphi \end{array} \quad \begin{array}{c} \vdots \\ \neg(\exists x.\neg B(x)) \vdash \varphi \end{array}}{\vdash \varphi} \text{(te)}$$

Si $B^I \subsetneq D_I$, dans la branche de gauche, on a la dérivation suivante :

$$\frac{\frac{\frac{\frac{}{\neg B(x), B(x) \vdash \neg B(x)} \text{(ax)}}{\neg B(x), B(x) \vdash \perp} \text{(}\neg e\text{)}}{\frac{\frac{\frac{}{\neg B(x), B(x) \vdash \perp} \text{(}\perp\text{)}}{\neg B(x), B(x) \vdash \forall y.B(y)} \text{(}\rightarrow i\text{)}}{\neg B(x) \vdash B(x) \rightarrow \forall y.B(y)} \text{(}\exists i\text{)}} \text{(W)}}{\frac{\frac{}{\exists x.\neg B(x) \vdash \exists x.\neg B(x)} \text{(ax)}}{\exists x.\neg B(x), \neg B(x) \vdash \varphi} \text{(}\exists e\text{)}} \text{(}\exists e\text{)}} \text{(W)}$$

$$\exists x.\neg B(x) \vdash \varphi$$

$$\frac{\frac{\frac{\overline{B(x), B(y) \vdash B(y)}^{(ax)}}{B(x) \vdash B(y) \rightarrow B(y)}^{(\rightarrow i)}}{B(x) \vdash \forall y. B(y) \rightarrow B(y)}^{(\forall i')}}{\vdash B(x) \rightarrow (\forall y. B(y) \rightarrow B(y))}^{(\rightarrow i)}$$

En fait, on peut se convaincre que cette variante de la déduction naturelle a besoin d'utiliser des α -renommages implicites pour être complète, sans quoi on n'arrive pas à prouver le séquent valide $B(x) \vdash (\forall x. B(x) \rightarrow B(x))$, car $B(x)$ fera partie de tous les contextes Γ des séquents que l'on peut trouver dans une recherche de preuve.

Remarque 10.5. Si l'on s'interdit d'utiliser l' α -renommage implicitement, mais si l'on se dote de la règle d'affaiblissement, on est en mesure de dériver ce séquent.

$$\frac{\frac{\frac{\frac{\overline{B(x) \vdash B(x)}^{(ax)}}{\vdash B(x) \rightarrow B(x)}^{(\rightarrow i)}}{\vdash \forall x. B(x) \rightarrow B(x)}^{(\forall i')}}{B(x) \vdash \forall x. B(x) \rightarrow B(x)}^{(W)}}{\vdash B(x) \rightarrow (\forall x. B(x) \rightarrow B(x))}^{(\rightarrow i)}$$

Cependant, l'admissibilité de la règle (W) (si on ne l'ajoute pas *a priori* au système de déduction) n'est pas évidente en déduction naturelle du premier ordre : par exemple, si on ajoute un ensemble Δ aux hypothèses d'un séquent $\Gamma \vdash \varphi$ où $x \notin \text{Libres}(\Gamma)$, pour pouvoir appliquer la règle ($\forall i'$) et déduire $\Gamma, \Delta \vdash \forall x. \varphi$, il faut aussi que $x \notin \text{Libres}(\Delta)$, ce qui peut nécessiter un α -renommage !

Dans le système de règles des figures 9.1 et 10.1, il est possible de démontrer l'admissibilité de l' α -renommage, et de justifier par la même occasion l'emploi d' α -renommages implicites dans les règles ($\exists e'$) et ($\forall i'$).

Commençons par étendre l' α -congruence aux ensembles : si $\varphi_i =_{\alpha} \psi_i$ pour tout $1 \leq i \leq n$, alors $\varphi_1, \dots, \varphi_n =_{\alpha} \psi_1, \dots, \psi_n$. Une substitution σ est *applicable* à un ensemble Γ si elle est applicable à chacune des formules de Γ . Si c'est le cas, on définit l'application d'une substitution σ à $\Gamma = \varphi_1, \dots, \varphi_n$ comme une application simultanée de σ à toutes les occurrences de formules dans Γ : $\Gamma\sigma \stackrel{\text{def}}{=} \varphi_1\sigma, \dots, \varphi_n\sigma$. Comme d'habitude, nous faisons un abus de notation et nous écrivons « $\Gamma\sigma$ » pour un ensemble $\Delta\sigma$ tel que $\Delta =_{\alpha} \Gamma$ et tel que σ soit applicable à Δ .

Les trois règles ci-dessous sont alors admissibles en déduction naturelle du premier ordre.

$$\frac{\Gamma \vdash \varphi}{\Delta \vdash \psi} (=_{\alpha}) \quad \frac{\Gamma \vdash \varphi}{\Gamma\sigma \vdash \varphi\sigma} (S) \quad \frac{\Gamma \vdash \varphi}{\Gamma, \Delta \vdash \varphi} (W)$$

où $\Gamma =_{\alpha} \Delta$ et $\varphi =_{\alpha} \psi$ où σ est une substitution

Lemme 10.6 (α -congruence syntaxique). *Soient $\Gamma =_{\alpha} \Delta$ deux ensembles finis de formules du premier ordre, et $\varphi =_{\alpha} \psi$ deux formules du premier ordre. Si $\Gamma \vdash_{\text{NK}} \varphi$, alors $\Delta \vdash_{\text{NK}} \psi$ par une dérivation de même taille.*

■ (MANCOSU, GALVAN et ZACH, 2021, lem. 3.18)

La démonstration du lemme d' α -congruence syntaxique est technique et nous n'allons pas la faire dans ces notes ; voir l'ouvrage de MANCOSU, GALVAN et ZACH, lem. 3.18 pour une démonstration d'un énoncé un peu plus faible mais qui suffit pour la suite.

Lemme 10.7 (substitution syntaxique). *Soit Γ un ensemble fini de formules du premier ordre,*

■ (MANCOSU, GALVAN et ZACH, 2021, thm. 3.20), (DAVID, NOUR et RAFFALLI, 2003, thm 1.6.8)

φ une formule du premier ordre, et σ une substitution. Si $\Gamma \vdash_{\text{NK}} \varphi$, alors $\Gamma\sigma \vdash_{\text{NK}} \varphi\sigma$ par une dérivation de même taille.

Voir (MANCOSU, GALVAN et ZACH, 2021, thm. 3.20) ou (DAVID, NOUR et RAFFALLI, 2003, thm 1.6.8) pour une démonstration du lemme 10.7 de substitution syntaxique.

Lemme 10.8 (admissibilité de l'affaiblissement). *Soient Γ et Δ deux ensembles finis de formules du premier ordre et φ une formule du premier ordre. Si $\Gamma \vdash_{\text{NK}} \varphi$ alors $\Gamma, \Delta \vdash_{\text{NK}} \varphi$ par une dérivation de même taille.*

Démonstration. On démontre le lemme par induction sur la taille des dérivations, en faisant une distinction de cas selon la dernière règle employée. Plusieurs cas ont déjà été évoqués pour le fragment propositionnel dans la section 9.2.1 ; ici on va traiter les cas où la dernière règle employée est $(\exists i)$ ou $(\exists e)$.

Pour la règle $(\exists i)$: Alors il existe une dérivation π de $\Gamma \vdash \varphi[t/x]$ où $t \in T(\mathcal{F}, X)$ qui permet de déduire $\Gamma \vdash \exists x.\varphi$, pour une taille totale $|\pi| + 1$. Par hypothèse d'induction, il existe une dérivation de $\Gamma, \Delta \vdash \varphi[t/x]$ de taille $|\pi|$, et en appliquant $(\exists i)$, on obtient une dérivation de $\Gamma, \Delta \vdash \exists x.\varphi$ de taille $|\pi| + 1$ comme désiré.

Pour la règle $(\exists e)$: Alors il existe deux dérivations π_1 de $\Gamma \vdash \exists x.\varphi$ et π_2 de $\Gamma, \varphi[y/x] \vdash \psi$ pour une variable $y \notin \text{Libres}(\Gamma, \exists x.\varphi, \psi)$ qui permettent de déduire $\Gamma \vdash \psi$, pour une taille totale $|\pi_1| + |\pi_2| + 1$.

Soit $z \notin \text{Libres}(\Gamma, \Delta, \exists x.\varphi, \psi)$; comme Γ et Δ sont finis, une telle variable existe bien dans X qui est supposé infini. Alors, par le lemme 10.7 de substitution syntaxique, il existe aussi une dérivation de $\Gamma[z/y], \varphi[y/x][z/y] \vdash \psi[z/y]$ de taille $|\pi_2|$.

Par hypothèse d'induction, il existe donc aussi des dérivations de $\Gamma, \Delta \vdash \exists x.\varphi$ et de $\Gamma[z/y], \Delta, \varphi[y/x][z/y] \vdash \psi[z/y]$ de tailles $|\pi_1|$ et $|\pi_2|$.

Comme $y \notin \text{Libres}(\Gamma, \psi)$, par l'équation (3.1), on a $\Gamma =_{\alpha} \Gamma[z/y]$ et $\psi =_{\alpha} \psi[z/y]$, et comme $z \notin \text{Libres}(\exists x.\varphi)$, par l'équation (3.2), on a $\varphi[y/x][z/y] =_{\alpha} \varphi[z/x]$. Par le lemme 10.6 d' α -congruence syntaxique, il existe donc une dérivation de $\Gamma, \Delta, \varphi[z/x] \vdash \psi$ de taille $|\pi_2|$. En appliquant la règle $(\exists e)$, on a alors une dérivation de $\Gamma, \Delta \vdash \psi$ de taille $|\pi_1| + |\pi_2| + 1$ comme désiré. \square

☞ Comme le problème de VALIDITÉ de la logique du premier ordre est indécidable (par le corollaire 2.16), la PROUVABILITÉ en déduction naturelle du premier ordre l'est donc aussi.

■ (DAVID, NOUR et RAFFALLI, 2003, prop. 2.5.13), (JAUME et al., 2020, thm. 5.1)

10.3. Correction et complétude. Comme dans le cas propositionnel, un séquent $\Gamma \vdash \varphi$ est valide si $\Gamma \models \varphi$, et les théorèmes 10.9 et 10.10 montrent qu'un séquent est valide si et seulement s'il est prouvable. En particulier, une formule du premier ordre φ est valide si et seulement si $\vdash_{\text{NK}} \varphi$.

Théorème 10.9 (correction de la déduction naturelle). *Soit Γ un ensemble fini de formules du premier ordre et φ une formule du premier ordre. Si $\Gamma \vdash_{\text{NK}} \varphi$, alors $\Gamma \models \varphi$.*

Démonstration. La démonstration se fait par induction sur les dérivations, et consiste à montrer comme dans le théorème 9.10 de correction propositionnelle de la déduction naturelle que chacune des règles est correcte. On va traiter ici le cas des règles $(\exists i)$ et $(\exists e)$; les autres cas sont similaires.

Règle $(\exists i)$: Supposons que la prémisse $\Gamma \vdash \varphi[t/x]$ soit valide pour un certain terme $t \in T(\mathcal{F}, X)$. Soient I une interprétation et ρ une valuation telles que $I, \rho \models \Gamma$. Alors $I, \rho \models \varphi[t/x]$. Par le lemme 3.5 de substitution, on a aussi $I, \rho[[t]_{\rho}^I/x] \models \varphi$. Autrement dit il existe $e \stackrel{\text{def}}{=} [[t]_{\rho}^I] \in D_I$ tel que $I, \rho[e/x] \models \varphi$, et donc $I, \rho \models \exists x.\varphi$ comme désiré.

Règle (\exists e) : Supposons que les deux prémisses $\Gamma \vdash \exists x.\varphi$ et $\Gamma, \varphi[y/x] \vdash \psi$ soient valides. Soient I une interprétation et ρ une valuation telles que $I, \rho \models \Gamma$ et montrons que $I, \rho \models \psi$.

Tout d'abord, par validité de la première prémisse, $I, \rho \models \exists x.\varphi$ donc il existe $e \in D_I$ tel que $I, \rho[e/x] \models \varphi$.

Par ailleurs, par validité de la seconde prémisse, pour tous I' et ρ' , si $I', \rho' \models \Gamma$ et $I', \rho' \models \varphi[y/x]$, alors $I', \rho' \models \psi$. Par le lemme 3.5 de substitution, cette deuxième condition est équivalente à demander $I', \rho' [[y]_{\rho'}^{I'}/x] \models \varphi$, soit encore $I', \rho'[\rho'(y)/x] \models \varphi$.

En particulier, pour $I' \stackrel{\text{def}}{=} I$ et $\rho' \stackrel{\text{def}}{=} \rho[e/y]$, on aura bien

- $I, \rho[e/y] \models \Gamma$ par la propriété 2.6 car $y \notin \text{Libres}(\Gamma)$ et $I, \rho \models \Gamma$, et
- $I, \rho[e/x] \models \varphi$ car $\rho'[\rho'(y)/x] = \rho[e/x]$.

On en déduit que $I, \rho[e/x] \models \psi$, et comme $y \notin \text{Libres}(\psi)$, encore par la propriété 2.6, on a aussi $I, \rho \models \psi$ comme désiré. \square

Théorème 10.10 (complétude de la déduction naturelle). *Soit Γ un ensemble fini de formules du premier ordre et φ une formule du premier ordre. Si $\Gamma \models \varphi$, alors $\Gamma \vdash_{\text{NK}} \varphi$.*

■ (DUPARC, 2015, thm. 439),
(DAVID, NOUR et RAFFALLI, 2003,
sec. 2.5.2), (JAUME et al., 2020,
sec. 5.2)

On ne donnera pas la démonstration du théorème 10.10, qui fait partie du programme complémentaire de fondements de l'informatique.

11. RECHERCHE AUTOMATIQUE DE PREUVE EN LOGIQUE PROPOSITIONNELLE

Résumé. Le *calcul des séquents propositionnel* est un système de déduction qui manipule des séquents $\vdash \Gamma$, où Γ est maintenant un multi-ensemble fini de formules propositionnelles sous forme normale négative. Un séquent $\vdash \Gamma$ pour lequel il existe une dérivation dans ce système de preuve est dit *prouvable*, ce qui est noté « $\vdash_{\text{LK}_0} \Gamma$ ».

On peut implémenter la *recherche de preuve* dans le calcul des séquents propositionnel : cela tient à la propriété 11.7 de branches linéaires, et de plus on peut faire une implémentation sans retours sur erreurs grâce au lemme 11.8 d'inversibilité syntaxique.

Un séquent $\vdash \Gamma$ est *valide*, si pour toute interprétation I , il existe une formule propositionnelle $\vartheta \in \text{dom}(\Gamma)$ du séquent telle que $I \models \vartheta$. Par le théorème 11.14 de *complétude propositionnelle du calcul des séquents*, si $\vdash \Gamma$ est valide, alors il est prouvable.

Enfin, une preuve en calcul des séquents propositionnel peut être traduite en une preuve en déduction naturelle propositionnelle (théorème 11.16), ce qui par complétude propositionnelle du calcul des séquents montre aussi la complétude propositionnelle de la déduction naturelle.

La *recherche de preuve* est le processus où l'on part d'un séquent que l'on cherche à démontrer, et où l'on vise à en trouver une dérivation ou à établir qu'il n'en existe aucune. Autrement dit, on cherche une dérivation « du bas vers le haut ».

Déjà dans le cas propositionnel de la section 9, la recherche de preuve en déduction naturelle est plutôt ardue. Ainsi, s'il est généralement indiqué d'appliquer les règles d'introduction autant que possible, l'emploi des règles d'élimination nécessite de « deviner » de nouvelles formules (à savoir φ dans $(\neg e)$, $(\wedge e_d)$ et $(\rightarrow e)$, ou δ dans $(\vee e)$ ou encore ψ dans $(\wedge e_g)$). Cela crée une infinité d'instanciations possibles de ces règles : la recherche a une « largeur » infinie.

De plus, le long d'une branche de recherche de preuve, on peut retomber sur un séquent que l'on a déjà rencontré ; un exemple très simple est le suivant, où l'on fait une recherche de preuve pour le séquent $\varphi \vdash \psi$, et on enchaîne inutilement les règles d'élimination et d'introduction de l'implication avant de se retrouver à rechercher encore une preuve de $\varphi \vdash \psi$:

$$\frac{\frac{\vdots}{\varphi \vdash \psi} \quad \frac{\varphi \vdash \varphi}{\varphi \vdash \varphi} \text{ (ax)}}{\varphi \vdash \varphi \rightarrow \psi} \text{ (}\rightarrow\text{i)} \quad \frac{\varphi \vdash \varphi}{\varphi \vdash \psi} \text{ (}\rightarrow\text{e)}$$

■ L'élimination de ces redondances s'appelle la normalisation ; voir (TROELSTRA et SCHWICHTENBERG, 2000, chap. 6) ou (MANCOSU, GALVAN et ZACH, 2021, chap. 4).

Ces redondances font que la recherche de preuve en déduction naturelle a aussi une « hauteur » infinie.

Pourtant, par correction et complétude propositionnelle de la déduction naturelle, un séquent est prouvable si et seulement s'il est valide, et le problème de VALIDITÉ est décidable dans le cas propositionnel. Il est donc possible de déterminer automatiquement l'existence d'une preuve en déduction naturelle propositionnelle. Ainsi, dans le système Coq de la section 9.4, au lieu de chercher manuellement une preuve de la loi de PEIRCE comme nous l'avons fait en section 9.4.1, on aurait pu utiliser la tactique `tauto` comme ci-dessous :

```
propositional.v
Lemma peirce : ((P -> Q) -> P) -> P.
tauto.
Qed.
```

En réalité, la tactique `tauto` de Coq fournit plus qu'une simple implémentation d'un algorithme pour le problème de VALIDITÉ propositionnelle. Cette tactique fait une recherche de preuve dans un système de preuve mieux adapté et traduit ensuite la dérivation trouvée (s'il en existe une) en une dérivation en déduction naturelle. Le reste de cette section est dédié à un tel système de preuve adapté pour la recherche de preuve, qui est un calcul des séquents, et à la traduction des dérivations du calcul des séquents vers la déduction naturelle, que nous verrons en section 11.4.

■ Voir (TROELSTRA et SCHWICHTENBERG, 2000, chap. 3) pour un panorama de différentes présentations du calcul des séquents; notre calcul, appelé « **GS3p** », y est introduit en sec. 3.6. Voir plus généralement (DUPARC, 2015, sec. 1.3.5), (GOUBAULT-LARRECQ et MACKIE, 1997, sec. 2.3.3), (DAVID, NOUR et RAFFALLI, 2003, chap. 5) pour des présentations plus habituelles du calcul des séquents sous la forme bilatère avec coupure.

11.1. * **Calcul des séquents propositionnel.** Le système de preuve que nous allons étudier pour la recherche de preuve est un système de calcul des séquents. Il y a quantité de variantes du calcul des séquents pour la logique classique propositionnelle. Le système que nous étudions ici est un calcul monolatère inversible sans coupure, qui a l'avantage de ne comporter que peu de règles et de se prêter particulièrement bien à la recherche de preuve.

Un *multi-ensemble* fini sur un ensemble S est formellement une fonction $m: S \rightarrow \mathbb{N}$ de domaine $\text{dom}(m) \stackrel{\text{def}}{=} \{e \in S \mid m(e) > 0\}$ fini; on peut aussi voir m comme une séquence finie de S^* modulo permutation.

Dans le calcul de la figure 11.1, un *séquent* est un multi-ensemble fini Γ de formules en forme normale négative, et est noté $\vdash \Gamma$. La virgule dénote l'union de multi-ensembles : par exemple, Γ, Δ, φ dénote le multi-ensemble avec $\Gamma(\varphi) + \Delta(\varphi) + 1$ occurrences de la formule propositionnelle φ , et $\Gamma(\psi) + \Delta(\psi)$ occurrences de $\psi \neq \varphi$. On note le séquent vide $\vdash \perp$, où $\perp(\varphi) \stackrel{\text{def}}{=} 0$ pour toute formule φ .

▲ Dans notre calcul des séquents, on travaille exclusivement avec des formules sous forme normale négative.

☞ Ce calcul ne contient pas ni la règle structurelle d'échange, qui est implicite parce que nous travaillons avec des multi-ensembles, ni la règle structurelle d'affaiblissement, qui est implicite dans la règle d'axiome (ax), ni la règle structurelle de contraction (qui sont toutes admissibles), ni la règle de coupure (qui peut être éliminée).

$$\frac{}{\vdash \Gamma, P, \neg P} \text{ (ax)} \quad \frac{\vdash \Gamma, \varphi \quad \vdash \Gamma, \psi}{\vdash \Gamma, \varphi \wedge \psi} (\wedge) \quad \frac{\vdash \Gamma, \varphi, \psi}{\vdash \Gamma, \varphi \vee \psi} (\vee)$$

FIGURE 11.1. Calcul des séquents propositionnel monolatère.

Chaque règle comprend une *formule principale* dans sa conclusion, indiquée en orange dans les règles de la figure 11.1. Un séquent $\vdash \Gamma$ est *prouvable*, noté $\vdash_{\text{LK}_0} \Gamma$, s'il en existe une dérivation dans le système de la figure 11.1.

Exemple 11.1. La loi de PEIRCE de l'exemple 7.2 est prouvable. La dérivation correspondante (avec la formule principale indiquée en orange à chaque étape) est :

$$\frac{\frac{\frac{}{\vdash \neg P, Q, P} \text{ (ax)}}{\vdash \neg P \vee Q, P} \text{ (}\vee\text{)}}{\vdash (\neg P \vee Q) \wedge \neg P, P} \text{ (}\wedge\text{)}}{\vdash ((\neg P \vee Q) \wedge \neg P) \vee P} \text{ (}\vee\text{)}$$

Exemple 11.2. Considérons la formule propositionnelle $(P \wedge P) \leftrightarrow P$. Sa forme normale négative est $(\neg P \vee \neg P \vee P) \wedge (\neg P \vee (P \wedge P))$. Cette formule est prouvable; une dérivation est :

$$\frac{\frac{\frac{\frac{}{\vdash \neg P, \neg P, P} \text{ (ax)}}{\vdash \neg P, \neg P \vee P} \text{ (}\vee\text{)}}{\vdash \neg P \vee \neg P \vee P} \text{ (}\vee\text{)}}{\vdash (\neg P \vee \neg P \vee P) \wedge (\neg P \vee (P \wedge P))} \text{ (}\wedge\text{)}}{\frac{\frac{\frac{}{\vdash \neg P, P} \text{ (ax)}}{\vdash \neg P, P \wedge P} \text{ (}\wedge\text{)}}{\vdash \neg P \vee (P \wedge P)} \text{ (}\vee\text{)}}{\vdash (\neg P \vee \neg P \vee P) \wedge (\neg P \vee (P \wedge P))} \text{ (}\wedge\text{)}}$$

Exemple 11.3. Considérons la formule propositionnelle $(P \wedge (Q \vee R) \rightarrow ((P \wedge Q) \vee (P \wedge R)))$. Sa forme normale négative est $\neg P \vee (\neg Q \wedge \neg R) \vee (P \wedge Q) \vee (P \wedge R)$. Cette formule est prouvable; une dérivation est :

$$\frac{\frac{\frac{\frac{\frac{}{\vdash \neg P, \neg Q, Q, R} \text{ (ax)}}{\vdash \neg P, \neg Q, \neg R, Q, R} \text{ (}\wedge\text{)}}{\vdash \neg P, \neg Q \wedge \neg R, Q, P} \text{ (ax)}}{\vdash \neg P, \neg Q \wedge \neg R, P \wedge Q, P \wedge R} \text{ (}\wedge\text{)}}{\vdash \neg P, \neg Q \wedge \neg R, (P \wedge Q) \vee (P \wedge R)} \text{ (}\vee\text{)}}{\vdash \neg P, (\neg Q \wedge \neg R) \vee (P \wedge Q) \vee (P \wedge R)} \text{ (}\vee\text{)}}{\vdash \neg P \vee (\neg Q \wedge \neg R) \vee (P \wedge Q) \vee (P \wedge R)} \text{ (}\vee\text{)}}$$

Exemple 11.4. Considérons les trois formules propositionnelles suivantes :

$$\varphi_A \stackrel{\text{def}}{=} B \rightarrow D, \quad \varphi_B \stackrel{\text{def}}{=} \neg B \rightarrow (\neg D \wedge \neg U), \quad \varphi_C \stackrel{\text{def}}{=} B \vee D \vee U.$$

On souhaite montrer que $B \wedge D$ est une conséquence logique de $\varphi_A \wedge \varphi_B \wedge \varphi_C$. Par le lemme 6.2 de déduction, $\varphi_A \wedge \varphi_B \wedge \varphi_C \models B \wedge D$ si et seulement si $(\varphi_A \wedge \varphi_B \wedge \varphi_C) \rightarrow B \wedge D$ est valide, autrement dit si et seulement si sa forme normale négative $\bar{\varphi}_A \vee \bar{\varphi}_B \vee \bar{\varphi}_C \vee (B \wedge D)$ est valide, où

$$\bar{\varphi}_A = B \wedge \neg D, \quad \bar{\varphi}_B = \neg B \wedge (D \vee U), \quad \bar{\varphi}_C = \neg B \wedge \neg D \wedge \neg U.$$

La dérivation ci-dessous montre que $\vdash_{\text{LK}_0} \bar{\varphi}_A \vee \bar{\varphi}_B \vee \bar{\varphi}_C \vee (B \wedge D)$, ce qui implique par le théorème 11.11 de correction propositionnelle du calcul des séquents que la formule est bien valide et donc que $B \wedge D$ est bien une conséquence logique de $\varphi_A \wedge \varphi_B \wedge \varphi_C$.

$$\frac{\frac{\frac{\frac{}{\vdash B, \neg B, \bar{\varphi}_C, D} \text{ (ax)}}{\vdash \bar{\varphi}_A, \neg B, \bar{\varphi}_C, D} \text{ (ax)}}{\vdash \bar{\varphi}_A, \neg B, \bar{\varphi}_C, B \wedge D} \text{ (}\wedge\text{)}}{\frac{\frac{\frac{\frac{}{\vdash \neg D, \neg B, \bar{\varphi}_C, D} \text{ (ax)}}{\vdash \bar{\varphi}_A, \neg B, \bar{\varphi}_C, D} \text{ (}\wedge\text{)}}{\vdash \bar{\varphi}_A, \bar{\varphi}_B, \bar{\varphi}_C, B \wedge D} \text{ (}\wedge\text{)}}{\vdash \bar{\varphi}_A, \bar{\varphi}_B, \bar{\varphi}_C \vee (B \wedge D)} \text{ (}\vee\text{)}}{\vdash \bar{\varphi}_A, \bar{\varphi}_B \vee \bar{\varphi}_C \vee (B \wedge D)} \text{ (}\vee\text{)}}{\vdash \bar{\varphi}_A, \bar{\varphi}_B \vee \bar{\varphi}_C \vee (B \wedge D)} \text{ (}\vee\text{)}}$$

où la dérivation π de $\vdash \bar{\varphi}_A, D, U, \bar{\varphi}_C, B \wedge D$ est

$$\frac{\frac{\text{échec}}{\vdash P, \neg Q} \quad \frac{\text{échec}}{\vdash P}}{\vdash P \vee \neg Q} (\vee) \quad \frac{\text{échec}}{\vdash P}}{\vdash (P \vee \neg Q) \wedge P} (\wedge)$$

Les deux séquents $\vdash P, \neg Q$ et $\vdash P$ sont des *séquents d'échec*, c'est-à-dire des séquents sur lequel aucune règle ne s'applique. Les deux branches de recherche de preuve sont des *branches d'échec*, car elles terminent sur des séquents d'échec.

Exemple 11.6. La formule propositionnelle $(P \vee \neg Q) \wedge (\neg P \wedge R)$ n'est pas prouvable. En effet, une recherche de preuve pour cette formule commence nécessairement par appliquer (\vee) :

$$\frac{\vdash P \vee Q, \neg P \wedge \neg R}{\vdash (P \vee \neg Q) \vee (\neg P \wedge R)} (\vee)$$

Il y a là un point de choix, selon qu'on utilise $P \vee Q$ ou $\neg P \wedge R$ comme formule principale dans le séquent $\vdash P \vee Q, \neg P \wedge \neg R$. Si on fait le premier choix, la recherche de preuve échoue puisqu'une des branches (celle de droite) termine sur le séquent d'échec $\vdash P, \neg Q, R$ et est donc une branche d'échec :

$$\frac{\frac{\text{échec}}{\vdash P, \neg Q, \neg P} (\text{ax}) \quad \frac{\text{échec}}{\vdash P, \neg Q, R}}{\vdash P, \neg Q, \neg P \wedge R} (\wedge) \quad \frac{\text{échec}}{\vdash P, \neg Q, R}}{\vdash P \vee \neg Q, \neg P \wedge R} (\vee)$$

Si on fait l'autre choix, celui de considérer $\neg P \wedge R$ comme principale, la recherche de preuve échoue aussi :

$$\frac{\frac{\text{échec}}{\vdash P, \neg Q, \neg P} (\text{ax}) \quad \frac{\text{échec}}{\vdash P, \neg Q, R}}{\vdash P \vee \neg Q, \neg P} (\vee) \quad \frac{\text{échec}}{\vdash P, \neg Q, R}}{\vdash P \vee \neg Q, R} (\vee)}{\vdash P \vee \neg Q, \neg P \wedge R} (\wedge)$$

Comme quel que soit le choix de formule principale, la recherche de preuve échoue, la formule n'est pas prouvable.

11.2.1. *Propriété de branches finies.* On peut montrer que les branches de recherche de preuve dans le calcul des séquents propositionnel de la figure 11.1 sont toutes finies, et même linéaires en la taille du séquent $\vdash \Gamma$ que l'on cherche à prouver.

On définit pour cela la *taille* d'un multi-ensemble Γ comme la somme des tailles de ses occurrences de formules $|\Gamma| \stackrel{\text{def}}{=} \sum_{\varphi \in \text{dom}(\Gamma)} |\varphi| \cdot \Gamma(\varphi)$, où la taille $|\varphi|$ d'une formule propositionnelle φ en forme normale négative est définie comme le nombre de nœuds étiquetés par \wedge ou \vee dans son arbre de syntaxe ; formellement :

$$|\ell| \stackrel{\text{def}}{=} 0, \quad |\varphi \vee \psi| = |\varphi \wedge \psi| \stackrel{\text{def}}{=} 1 + |\varphi| + |\psi|.$$

Intuitivement, la taille d'une formule propositionnelle est le nombre de nœuds étiquetés par « \vee » ou « \wedge » de son arbre de syntaxe abstraite, et la taille d'un séquent est la somme des tailles de ses formules.

Propriété 11.7 (branches linéaires). *Soit $\vdash \Gamma$ un séquent propositionnel. Alors toutes les branches d'une recherche de preuve pour $\vdash \Gamma$ sont de longueur au plus $|\Gamma|$.*

Démonstration. On peut vérifier que pour chacune des règles (ax), (\vee) et (\wedge), la taille de chaque séquent prémisses est strictement inférieure à celle du séquent conclusion. \square

Voici comment la recherche de preuve pourrait s'écrire en pseudo-code, qui pour un séquent courant $\vdash \Gamma$ essaye tour à tour d'utiliser chaque formule $\varphi \in \text{dom}(\Gamma)$ en guise de formule principale.

```

Fonction prouvable( $\vdash \Gamma$ )
1   $v := F$ 
2  pour tous les  $\varphi \in \text{dom}(\Gamma)$  faire
3    si  $\varphi = \ell$  et  $\bar{\ell} \in \text{dom}(\Gamma)$  alors
4       $\lfloor$  retourner  $V$ 
5    si  $\varphi = \varphi' \vee \psi$  alors
6       $\Delta := \Gamma \setminus \varphi$ 
7       $v := v$  ou prouvable( $\vdash \Delta, \varphi', \psi$ )
8    si  $\varphi = \varphi' \wedge \psi$  alors
9       $\Delta := \Gamma \setminus \varphi$ 
10      $v := v$  ou (prouvable( $\vdash \Delta, \varphi'$ ) et prouvable( $\vdash \Delta, \psi$ ))
11 retourner  $v$ 

```

Par la propriété 11.7 de branches linéaires, cet algorithme termine bien : la profondeur des appels récursifs est bornée par $|\Gamma|$.

11.2.2. *Inversibilité et algorithme de recherche de preuve.* Nous allons maintenant voir que le choix d'une formule principale n'est pas important dans notre calcul des séquents, et que l'on peut donc considérablement simplifier le pseudo-code de prouvable. On peut commencer par observer que, dans nos deux exemples précédents, la recherche de preuve réussit pour nos deux choix de formule principale :

$$\begin{array}{c}
 \frac{}{\vdash \neg P, Q, R, P} \text{ (ax)} \\
 \frac{}{\vdash \neg P, Q, R \vee P} \text{ (}\vee\text{)} \\
 \frac{}{\vdash \neg P \vee Q, R \vee P} \text{ (}\vee\text{)}
 \end{array}
 \qquad
 \begin{array}{c}
 \frac{}{\vdash \neg P, Q, R, P} \text{ (ax)} \\
 \frac{}{\vdash \neg P \vee Q, R, P} \text{ (}\vee\text{)} \\
 \frac{}{\vdash \neg P \vee Q, R \vee P} \text{ (}\vee\text{)}
 \end{array}$$

Une règle de déduction est *syntactiquement inversible* si, quand il existe une dérivation de son séquent conclusion, alors il existe une dérivation de chacun de ses séquents prémisses. La règle (ax) est bien sûr syntactiquement inversible puisqu'elle n'a aucune prémisses. Mais ce qui est plus intéressant, c'est que toutes les autres règles de la figure 11.1 sont elles aussi syntactiquement inversibles. Cela signifie que dans une recherche de preuve, on peut appliquer ces règles de manière gloutonne sans faire de *backtracking* – donc sans avoir à mémoriser de points de choix – et que si l'on arrive à un séquent pour lequel aucune règle ne s'applique, alors c'est qu'il n'y avait pas de preuve.

Lemme 11.8 (inversibilité syntaxique). *Les règles du calcul des séquents propositionnel sont syntactiquement inversibles.*

Démonstration. Comme déjà mentionné, la règle (ax) est syntactiquement inversible par définition puisqu'elle n'a pas de prémisses.

Pour la règle (\vee) : supposons qu'il existe une dérivation π du séquent $\vdash \Gamma, \varphi \vee \psi$. On montre par récurrence sur la profondeur de π que $\vdash_{\text{LK}_0} \Gamma, \varphi, \psi$.

- Si π se termine par (\vee) où $\varphi \vee \psi$ est principale, alors évidemment il existe une sous-dérivation de π pour sa prémisse $\vdash \Gamma, \varphi, \psi$.
- Sinon π se termine par une règle (R) où $\varphi \vee \psi$ n'est pas principale. Par inspection des règles, on est nécessairement dans une situation

$$\frac{\begin{array}{c} \pi_1 \\ \vdots \\ \vdash \Gamma_1, \varphi \vee \psi \end{array} \quad \cdots \quad \begin{array}{c} \pi_k \\ \vdots \\ \vdash \Gamma_k, \varphi \vee \psi \end{array}}{\vdash \Gamma, \varphi \vee \psi} \text{ (R)}$$

où $0 \leq k \leq 2$ ($k = 0$ correspondant au cas de la règle (ax)). Pour tout $1 \leq i \leq k$, par hypothèse de récurrence sur π_i , il existe des dérivations π'_i de $\vdash \Gamma_i, \varphi, \psi$. On a donc la dérivation

$$\frac{\begin{array}{c} \pi'_1 \\ \vdots \\ \vdash \Gamma_1, \varphi, \psi \end{array} \quad \cdots \quad \begin{array}{c} \pi'_k \\ \vdots \\ \vdash \Gamma_k, \varphi, \psi \end{array}}{\vdash \Gamma, \varphi, \psi} \text{ (R)}$$

Pour la règle (\wedge) : supposons qu'il existe une dérivation π du séquent $\vdash \Gamma, \varphi \wedge \psi$. On montre par récurrence sur la profondeur de π que $\vdash_{\text{LK}_0} \Gamma, \varphi$ et $\vdash_{\text{LK}_0} \Gamma, \psi$.

- Si π se termine par (\wedge) où $\varphi \wedge \psi$ est principale, alors évidemment il existe des sous-dérivations de π pour chacune de ses prémisses $\vdash \Gamma, \varphi$ et $\vdash \Gamma, \psi$.
- Sinon π se termine par une règle (R) où $\varphi \wedge \psi$ n'est pas principale. Par inspection des règles, on est nécessairement dans une situation

$$\frac{\begin{array}{c} \pi_1 \\ \vdots \\ \vdash \Gamma_1, \varphi \wedge \psi \end{array} \quad \cdots \quad \begin{array}{c} \pi_k \\ \vdots \\ \vdash \Gamma_k, \varphi \wedge \psi \end{array}}{\vdash \Gamma, \varphi \wedge \psi} \text{ (R)}$$

où $0 \leq k \leq 2$ ($k = 0$ correspondant au cas de la règle (ax)). Pour tout $1 \leq i \leq k$, par hypothèse de récurrence sur π_i , il existe des dérivations π'_i et π''_i de $\vdash \Gamma_i, \varphi$ et de $\vdash \Gamma_i, \psi$. On a donc les dérivations

$$\frac{\begin{array}{c} \pi'_1 \\ \vdots \\ \vdash \Gamma_1, \varphi \end{array} \quad \cdots \quad \begin{array}{c} \pi'_k \\ \vdots \\ \vdash \Gamma_k, \varphi \end{array}}{\vdash \Gamma, \varphi} \text{ (R)} \quad \text{et} \quad \frac{\begin{array}{c} \pi''_1 \\ \vdots \\ \vdash \Gamma_1, \psi \end{array} \quad \cdots \quad \begin{array}{c} \pi''_k \\ \vdots \\ \vdash \Gamma_k, \psi \end{array}}{\vdash \Gamma, \psi} \text{ (R)}$$

□

Exemple 11.9. Reprenons la recherche de preuve de l'exemple 11.6. Considérons l'application de la règle (\vee) sur la formule principale $P \vee \neg Q$ dans

$$\frac{\frac{\frac{\vdash P, \neg Q, \neg P}{\vdash P, \neg Q, \neg P} \text{ (ax)} \quad \text{échec}}{\vdash P, \neg Q, \neg P \wedge R} \text{ (\wedge)}}{\vdash P \vee \neg Q, \neg P \wedge R} \text{ (\vee)}$$

Au moins une prémisse de cette règle n'est pas prouvable, à savoir $\vdash P, \neg Q, \neg P \wedge R$. Par le lemme 11.8 d'inversibilité syntaxique, la conclusion $\vdash P \vee \neg Q, \neg P \wedge R$ n'est pas prouvable. Il est donc inutile d'essayer de choisir la formule $\neg P \wedge R$ comme principale,

puisque la recherche de preuve échouera aussi dans ce cas (comme on a pu le vérifier dans l'exemple 11.6).

De plus, quand on a comme ici le choix entre formule \vee et une formule \wedge comme principale, il vaut mieux choisir la formule \vee qui mène à un arbre de recherche de preuve plus petit (comme on a aussi pu le voir dans l'exemple 11.6).

Grâce au lemme 11.8 d'inversibilité syntaxique, la recherche de preuve peut s'implémenter très aisément par un programme récursif qui travaille en temps au pire exponentiel et dont voici le pseudo-code.

```

Fonction prouvable( $\vdash \Gamma$ )
1 si  $\vdash \Gamma = \vdash \Delta, P, \neg P$  alors
2   retourner  $\vee$ 
3 si  $\vdash \Gamma = \vdash \Delta, \varphi \vee \psi$  alors
4   retourner prouvable( $\vdash \Delta, \varphi, \psi$ )
5 si  $\vdash \Gamma = \vdash \Delta, \varphi \wedge \psi$  alors
6   retourner prouvable( $\vdash \Delta, \varphi$ ) et prouvable( $\vdash \Delta, \psi$ )
7 retourner  $\text{F}$ 

```

Corollaire 11.10. PROUVABILITÉ est dans coNP .

Démonstration. On implémente la recherche d'une preuve pour $\vdash \Gamma$ dans le calcul des séquents propositionnel à l'aide d'une machine de TURING alternante. Celle-ci a des états existentiels pour choisir quelle règle appliquer au séquent courant et des états universels pour choisir quelle prémisse de cette règle on va continuer à explorer. Par la propriété 11.7, toutes les branches sont finies et de longueur linéaire en $|\Gamma|$; de plus vérifier qu'une règle est applicable au séquent courant est aussi en temps polynomial. Cela fournit en première analyse un algorithme en $\text{AP} = \text{PSPACE}$.


Cependant, par le lemme 11.8, tous les choix de règle à appliquer se valent, donc les états existentiels peuvent être remplacés par un choix déterministe. L'algorithme résultant est alors en coNP . \square


11.3. * **Correction et complétude du calcul des séquents propositionnel.** Un séquent $\vdash \Gamma$ est *satisfait* par une interprétation I , ce qu'on écrit $I \models \Gamma$, s'il existe une formule témoin $\vartheta \in \text{dom}(\Gamma)$ telle que $I \models \vartheta$. On dit qu'un séquent $\vdash \Gamma$ est *valide* et on écrit $\models \Gamma$ si pour toute interprétation I , $I \models \Gamma$.

Notons que ces définitions généralisent bien les notions de satisfiabilité et de validité des formules propositionnelles. En effet, si $\Gamma = \varphi$ une formule propositionnelle en forme normale négative, par la définition ci-dessus, $\vdash \varphi$ est satisfait par une interprétation I si et seulement s'il existe une formule témoin $\vartheta \in \text{dom}(\Gamma)$ telle que $I \models \vartheta$; comme $\text{dom}(\Gamma) = \{\varphi\}$, le seul choix possible pour ϑ est $\vartheta = \varphi$, et donc I satisfait $\vdash \varphi$ si et seulement si $I \models \varphi$.

La correction et la complétude du calcul des séquents montrent qu'un séquent est prouvable si et seulement s'il est valide. La correction se montre par une simple induction sur les dérivations en calcul des séquents.

Théorème 11.11 (correction propositionnelle du calcul des séquents). *Soit Γ un multi-ensemble fini de formules propositionnelles sous forme normale négative. Si $\vdash_{\text{LK}_0} \Gamma$, alors $\models \Gamma$.*

 Par la correction et complétude du calcul des séquents propositionnel, on a une réduction à VALIDITÉ et donc une preuve aisée que PROUVABILITÉ est dans coNP . L'intérêt ici est qu'on obtient cette même borne de complexité directement par la recherche de preuve.

 Cette notation ne correspond pas à la notion habituelle de satisfaction d'un ensemble de formules : ici on considère la disjonction des formules de Γ au lieu de leur conjonction.

Démonstration. On procède par induction structurale sur une dérivation de $\vdash \Gamma$, en montrant pour chaque règle du calcul des séquents que si les prémisses sont valides, alors la conclusion l'est aussi.

Pour (ax) : alors $\Gamma = \Gamma', P, \neg P$ pour une formule φ . Pour toute interprétation I , soit $I \models P$, soit $I \models \neg P$; dans tous les cas $I \models \Gamma$.

Pour (\vee) : alors $\Gamma = \Gamma', \varphi \vee \psi$ où $\vdash_{\text{LK}_0} \Gamma', \varphi, \psi$. Soit I une interprétation quelconque; alors par hypothèse d'induction $I \models \Gamma', \varphi, \psi$. Il existe donc une formule témoin $\vartheta \in \text{dom}(\Gamma') \cup \{\varphi, \psi\}$ telle que $I \models \vartheta$. Si $\vartheta \in \{\varphi, \psi\}$, alors $I \models \varphi \vee \psi$, sinon $\vartheta \in \text{dom}(\Gamma')$ et alors $I \models \Gamma'$; dans tous les cas $I \models \Gamma$.

Pour (\wedge) : alors $\Gamma = \Gamma', \varphi \wedge \psi$ où $\vdash_{\text{LK}_0} \Gamma', \varphi$ et $\vdash_{\text{LK}_0} \Gamma', \psi$. Soit I une interprétation quelconque; alors par hypothèse d'induction $I \models \Gamma', \varphi$ et $I \models \Gamma', \psi$. Il existe donc une formule témoin $\vartheta \in \text{dom}(\Gamma') \cup \{\varphi\}$ telle que $I \models \vartheta$ et une formule témoin $\vartheta' \in \text{dom}(\Gamma') \cup \{\psi\}$ telle que $I \models \vartheta'$. Si $\vartheta \in \text{dom}(\Gamma')$ ou $\vartheta' \in \text{dom}(\Gamma')$, alors $I \models \Gamma'$. Sinon, $\vartheta = \varphi$ et $\vartheta' = \psi$ et donc $I \models \varphi \wedge \psi$. Dans tous les cas $I \models \Gamma$. \square

Pour la complétude du calcul des séquents, c'est-à-dire montrer que si un séquent est valide, alors il est prouvable, rappelons qu'une *branche d'échec* est une branche de recherche de preuve finie $\vdash \Gamma_0, \vdash \Gamma_1, \dots, \vdash \Gamma_n$ où aucune règle ne s'applique au séquent $\vdash \Gamma_n$; on appelle un tel séquent un *séquent d'échec*.

Propriété 11.12 (contre-modèle des séquents d'échec). *Soit $\vdash \Gamma$ un séquent d'échec. Alors il en existe un contre-modèle, c'est-à-dire une interprétation I telle que $I \not\models \Gamma$.*

Démonstration. Par définition d'un séquent d'échec, aucune règle ne s'applique à $\vdash \Gamma$. Par suite,

- comme ni (\vee) ni (\wedge) ne s'applique, $\text{dom}(\Gamma)$ ne contient que des littéraux : $\text{dom}(\Gamma) = \{\ell_1, \dots, \ell_k\}$;
- comme (ax) ne s'applique pas non plus, $\text{dom}(\Gamma)$ ne contient pas à la fois un littéral et sa négation.

Il existe donc une interprétation I telle que $I \not\models \ell_j$ pour tout $1 \leq j \leq k$, c'est-à-dire telle que $I \not\models \ell_1, \dots, \ell_k$ et donc $I \not\models \Gamma$. \square

Par exemple, les branches d'échec de l'exemple 11.6 se terminent sur le séquent d'échec $\vdash P, \neg Q, R$, et toute interprétation qui étend $[F/P, V/Q, F/R]$ en est un contre-modèle.

L'idée de la preuve de complétude qui va suivre est que l'existence d'un contre-modèle pour le séquent d'échec $\vdash \Gamma_n$ d'une branche d'échec $\vdash \Gamma_0, \vdash \Gamma_1, \dots, \vdash \Gamma_n$ implique l'existence d'un contre-modèle pour le séquent $\vdash \Gamma_0$. Nous nous appuyons pour cela sur une version « sémantique » de l'inversibilité de notre calcul des séquents. On dit pour cela qu'une règle est *sémantiquement inversible* si, quand sa conclusion est valide, alors chacune de ses prémisses est aussi valide.

Lemme 11.13 (inversibilité sémantique). *Les règles du calcul des séquents propositionnel sont sémantiquement inversibles. Plus précisément, si une interprétation I est un contre-modèle d'une prémisses, alors c'est aussi un contre-modèle de la conclusion.*

Démonstration.

Pour la règle (ax) : comme elle n'a pas de prémisses, elle est bien sémantiquement inversible.

Pour la règle (\vee) : supposons $I \not\models \Gamma, \varphi, \psi$, donc $I \not\models \Gamma, I \not\models \varphi$ et $I \not\models \psi$. On en déduit que $I \not\models \varphi \vee \psi$, et comme $I \not\models \Gamma$, que $I \not\models \Gamma, \varphi \vee \psi$.

Pour la règle (\wedge) : supposons que $I \not\models \Gamma, \varphi$ (le cas où $I \not\models \Gamma, \psi$ est similaire). Donc $I \not\models \Gamma$ et $I \not\models \varphi$. On en déduit que $I \not\models \varphi \wedge \psi$ et donc, puisque $I \not\models \Gamma$, que $I \not\models \Gamma, \varphi \wedge \psi$. \square

Dans l'exemple 11.6, toute interprétation I qui étend $[F/P, V/Q, F/R]$ est un contre-modèle du séquent $\vdash P, \neg Q, R$, et par applications successives du lemme 11.13 d'inversibilité sémantique, on en déduit que I est un contre-modèle du séquent initial $\vdash (P \vee \neg Q) \vee (\neg P \wedge R)$, qui n'est donc pas valide. C'est ce raisonnement que nous généralisons pour démontrer le théorème de complétude propositionnelle du calcul des séquents.

Théorème 11.14 (complétude propositionnelle du calcul des séquents). *Soit Γ un ensemble fini de formules propositionnelles sous forme normale négative. Si $\models \Gamma$, alors $\vdash_{\text{LK}_0} \Gamma$.*

Démonstration. On procède par contraposition : on suppose $\vdash \Gamma$ non prouvable, et on montre que $\vdash \Gamma$ n'est pas valide, en exhibant un contre-modèle du séquent $\vdash \Gamma$, c'est-à-dire une interprétation I telle que $I \not\models \Gamma$.

On commence par observer que si $\vdash \Gamma$ n'est pas prouvable, alors il existe une branche d'échec. En effet, par la propriété 11.7, les branches de recherche de preuve sont finies, donc elles terminent toutes soit par un séquent d'échec, soit par une instance de la règle d'axiome. Mais si $\vdash \Gamma$ avait une recherche de preuve où toutes les branches se terminaient par une instance de la règle d'axiome, alors cette recherche aurait construit une dérivation et donc $\vdash \Gamma$ serait prouvable.

Soit donc $\vdash \Gamma_0, \dots, \vdash \Gamma_n$ une branche d'échec où $\vdash \Gamma_0 = \vdash \Gamma$ et où $\vdash \Gamma_n$ est un séquent d'échec. Par la propriété 11.12 de contre-modèle des séquents d'échec, il existe un contre-modèle I de $\vdash \Gamma_n$, tel que $I \not\models \Gamma_n$.

On montre par récurrence décroissante sur $n \geq i \geq 0$ que $I \not\models \Gamma_i$. Pour le cas de base au rang $i = n$, on avait bien choisi I tel que $I \not\models \Gamma_n$. Pour l'étape de récurrence au rang $i - 1$, on sait par hypothèse de récurrence que $I \not\models \Gamma_i$, et par le lemme 11.13 d'inversibilité sémantique, $I \not\models \Gamma_{i-1}$. Pour conclure, on a montré qu'il existait un contre-modèle I de $\vdash \Gamma_0$, c'est-à-dire de $\vdash \Gamma$, qui n'est donc pas valide. \square

☞ On peut remarquer que, puisque le calcul des séquents est correct et complet, l'inversibilité syntaxique et l'inversibilité sémantique sont deux propriétés équivalentes : la première parle de prouvabilité, tandis que la seconde parle de validité.

■ On trouve de telles traductions entre calcul des séquents et déduction naturelle dans (DAVID, NOUR et RAFFALLI, 2003, sec. 5.3), (GOUBAULT-LARRECQ et MACKIE, 1997, sec. 3.2.3), (TROELSTRA et SCHWICHTENBERG, 2000, sec. 3.3). Le système COQ implémente la tactique `tauto` à l'aide d'une traduction depuis un calcul de séquents intuitionniste sans contraction dû à DYCKHOFF (1992).

11.4. * Traduction du calcul des séquents propositionnel en déduction naturelle.

Revenons à la motivation initiale de cette section, à savoir la recherche de preuve pour la déduction naturelle. L'idée ci-dessous va être de traduire une dérivation dans le calcul des séquents propositionnel de la figure 11.1 en une dérivation en déduction naturelle définie dans la figure 9.1. Une traduction dans la direction inverse, depuis des dérivations pour des formules sous forme normale négative en déduction naturelle vers le calcul des séquents, serait elle aussi possible mais nécessiterait l'ajout d'une règle de coupure à notre calcul des séquents.

11.4.1. *Règles d'introduction à gauche en déduction naturelle.* L'essentiel de la traduction du calcul des séquents en déduction naturelle réside dans l'admissibilité des trois règles d'insertion à gauche ci-dessous.

Proposition 11.15 (admissibilité des règles d'introduction à gauche). *Les trois règles ci-*

■ Voir (DAVID, NOUR et RAFFALLI, 2003, props. 1.3.19, 1.3.20 et 1.3.22).

dessous sont admissibles en déduction naturelle propositionnelle.

$$\frac{}{\Gamma, \varphi, \neg\varphi \vdash \perp} (\neg g) \quad \frac{\Gamma, \varphi, \psi \vdash \delta}{\Gamma, \varphi \wedge \psi \vdash \delta} (\wedge g) \quad \frac{\Gamma, \varphi \vdash \delta \quad \Gamma, \psi \vdash \delta}{\Gamma, \varphi \vee \psi \vdash \delta} (\vee g)$$

Démonstration. On a en effet les dérivations suivantes dans le système de la figure 9.1.

$$\frac{\frac{\Gamma, \varphi, \neg\varphi \vdash \neg\varphi}{\Gamma, \varphi, \neg\varphi \vdash \perp} (\text{ax}) \quad \frac{\Gamma, \varphi, \neg\varphi \vdash \varphi}{\Gamma, \varphi, \neg\varphi \vdash \perp} (\text{ax})}{\Gamma, \varphi, \neg\varphi \vdash \perp} (\neg e)$$

$$\frac{\frac{\Gamma, \varphi, \psi \vdash \delta}{\Gamma, \varphi \wedge \psi, \varphi, \psi \vdash \delta} (\text{w}) \quad \frac{\Gamma, \varphi \wedge \psi, \varphi \vdash \varphi \wedge \psi}{\Gamma, \varphi \wedge \psi, \varphi \vdash \psi} (\wedge e) \quad \frac{\Gamma, \varphi \wedge \psi \vdash \varphi \wedge \psi}{\Gamma, \varphi \wedge \psi \vdash \varphi} (\wedge e)}{\frac{\Gamma, \varphi \wedge \psi, \varphi \vdash \delta}{\Gamma, \varphi \wedge \psi \vdash \delta} (\text{cut})} (\text{cut})$$

$$\frac{\frac{\Gamma, \varphi \vee \psi \vdash \varphi \vee \psi}{\Gamma, \varphi \vee \psi \vdash \varphi \vee \psi} (\text{ax}) \quad \frac{\Gamma, \varphi \vdash \delta}{\Gamma, \varphi \vee \psi, \varphi \vdash \delta} (\text{w}) \quad \frac{\Gamma, \psi \vdash \delta}{\Gamma, \varphi \vee \psi, \psi \vdash \delta} (\text{w})}{\Gamma, \varphi \vee \psi \vdash \delta} (\vee e) \quad \square$$

11.4.2. *Traduction des preuves de formules sous forme normale négative.* Les trois règles d'introduction à gauche en déduction naturelle fournissent une démonstration immédiate de la traduction suivante du calcul des séquents en déduction naturelle.

Théorème 11.16. *Soit Γ un ensemble fini de formules sous forme normale négative. Si $\vdash_{\text{LK}_0} \Gamma$, alors $\bar{\Gamma} \vdash_{\text{NK}_0} \perp$.*

Démonstration. On procède par récurrence sur la hauteur d'une dérivation de $\vdash \Gamma$ dans le calcul des séquents propositionnel. On fait une analyse de cas selon la dernière règle employée, et on montre à chaque fois comment traduire cette dérivation en une dérivation de $\bar{\Gamma} \vdash \perp$ en déduction naturelle propositionnelle en utilisant l'admissibilité des règles d'introduction à gauche.

Cas de (ax) : alors $\Gamma = \Delta, P, \neg P$ et $\bar{\Gamma} = \bar{\Delta}, \neg P, P$ et on a la dérivation en déduction naturelle

$$\frac{}{\bar{\Delta}, \neg P, P \vdash \perp} (\neg g)$$

Cas de (\wedge) : alors $\Gamma = \Delta, \varphi \wedge \psi$ découle des prémisses $\vdash_{\text{LK}_0} \Delta, \varphi$ et $\vdash_{\text{LK}_0} \Delta, \psi$, et $\bar{\Gamma} = \bar{\Delta}, \bar{\varphi} \vee \bar{\psi}$ et on a la dérivation en déduction naturelle

$$\frac{\begin{array}{c} \text{h.i.} \\ \vdots \\ \bar{\Delta}, \bar{\varphi} \vdash \perp \end{array} \quad \begin{array}{c} \text{h.i.} \\ \vdots \\ \bar{\Delta}, \bar{\psi} \vdash \perp \end{array}}{\bar{\Delta}, \bar{\varphi} \vee \bar{\psi} \vdash \perp} (\vee g)$$

Cas de (\vee) : alors $\Gamma = \Delta, \varphi \vee \psi$ découle de la prémisse $\vdash_{\text{LK}_0} \Delta, \varphi, \psi$, et $\bar{\Gamma} = \bar{\Delta}, \bar{\varphi} \wedge \bar{\psi}$ et on a la dérivation en déduction naturelle

$$\frac{\begin{array}{c} \text{h.i.} \\ \vdots \\ \bar{\Delta}, \bar{\varphi}, \bar{\psi} \vdash \perp \end{array}}{\bar{\Delta}, \bar{\varphi} \wedge \bar{\psi} \vdash \perp} (\wedge g)$$

On a bien montré dans tous les cas que $\bar{\Gamma} \vdash_{\text{NK}_0} \perp$. □

11.4.3. *Règles de mise sous forme normale négative en déduction naturelle.* D'après le théorème 11.14 de complétude propositionnelle du calcul des séquents, une formule propositionnelle valide est prouvable en calcul des séquents, et par le théorème 11.16 on obtient une dérivation en déduction naturelle correspondante. Cela fournit presque une démonstration du théorème de complétude propositionnelle de la déduction naturelle, si ce n'est que notre calcul des séquents travaille exclusivement sur des formules sous forme normale négative, tandis que la déduction naturelle permet de traiter n'importe quelle formule propositionnelle. La pièce manquante à ce puzzle est de montrer que l'on peut « prouver » la mise sous forme normale négative au sein de la déduction naturelle.

Rappelons que $\bar{\varphi} \stackrel{\text{def}}{=} \text{nnf}(\neg\varphi)$. Nous allons montrer que les deux variantes suivantes de (ax) et (\neg) sont admissibles :

$$\frac{}{\varphi \vdash \text{nnf}(\varphi)} \text{ (nnf)} \quad \frac{}{\neg\bar{\varphi} \vdash \varphi} \text{ (nnf)}$$

Proposition 11.17 (admissibilité de la forme normale négative). *Les règles (nnf) et $\overline{\text{nnf}}$ sont admissibles en déduction naturelle propositionnelle.*

Démonstration. On procède par induction sur la formule propositionnelle φ .

Cas de P : alors $\text{nnf}(P) = P$ et $\bar{P} = \neg P$ et on a $P \vdash_{\text{NK}_0} P$ par (ax) et $\neg\neg P \vdash_{\text{NK}_0} P$ par (\neg).

Cas de $\neg\varphi$: alors $\text{nnf}(\neg\varphi) = \bar{\varphi}$ et $\overline{\neg\varphi} = \text{nnf}(\varphi)$. Pour (nnf), on a la dérivation

$$\frac{\frac{\frac{}{\neg\neg\bar{\varphi} \vdash \bar{\varphi}}{(\neg)} \quad \frac{}{\neg\bar{\varphi} \vdash \bar{\varphi}}{(\text{W})}}{\neg\varphi, \neg\neg\bar{\varphi} \vdash \bar{\varphi}} \quad \frac{\text{h.i. } \overline{\text{nnf}} \quad \vdots \quad \frac{}{\neg\bar{\varphi} \vdash \varphi}}{(\text{contra})}}{\neg\varphi \vdash \bar{\varphi}} \text{ (cut)}$$

Pour $\overline{\text{nnf}}$, on a la dérivation

$$\frac{\text{h.i. (nnf)} \quad \vdots \quad \frac{}{\varphi \vdash \text{nnf}(\varphi)}}{\neg\text{nnf}(\varphi) \vdash \neg\varphi} \text{ (contra)}$$

Cas de $\varphi \vee \psi$: alors $\text{nnf}(\varphi \vee \psi) = \text{nnf}(\varphi) \vee \text{nnf}(\psi)$ et $\overline{\varphi \vee \psi} = \bar{\varphi} \wedge \bar{\psi}$. On a alors la dérivation de (nnf) suivante

$$\frac{\frac{\text{h.i. (nnf)} \quad \vdots \quad \frac{}{\varphi \vdash \text{nnf}(\varphi)}}{\varphi \vdash \text{nnf}(\varphi) \vee \text{nnf}(\psi)} \text{ (v}_g\text{)} \quad \frac{\text{h.i. (nnf)} \quad \vdots \quad \frac{}{\psi \vdash \text{nnf}(\psi)}}{\psi \vdash \text{nnf}(\varphi) \vee \text{nnf}(\psi)} \text{ (v}_g\text{)}}{\frac{\varphi \vee \psi \vdash \varphi \vee \psi \text{ (ax)} \quad \frac{\varphi \vee \psi, \varphi \vdash \text{nnf}(\varphi) \vee \text{nnf}(\psi)}{\varphi \vee \psi, \varphi \vdash \text{nnf}(\varphi) \vee \text{nnf}(\psi)} \text{ (W)} \quad \frac{\varphi \vee \psi, \psi \vdash \text{nnf}(\varphi) \vee \text{nnf}(\psi)}{\varphi \vee \psi, \psi \vdash \text{nnf}(\varphi) \vee \text{nnf}(\psi)} \text{ (W)}}{\varphi \vee \psi \vdash \text{nnf}(\varphi) \vee \text{nnf}(\psi)} \text{ (v}_e\text{)}$$

Pour $\overline{\text{nnf}}$, on a une dérivation assez similaire

$$\begin{array}{c}
\text{exemple 9.7} \\
\vdots \\
\frac{\frac{\text{h.i. } (\overline{\text{nnf}})}{\vdots} \quad \frac{\overline{\neg\varphi} \vdash \varphi}{\overline{\neg\varphi} \vdash \varphi \vee \psi} (\vee_{ig})}{\overline{\neg(\varphi \wedge \overline{\psi})}, \overline{\neg\varphi} \vdash \varphi \vee \psi} (\vee) \quad \frac{\frac{\text{h.i. } (\overline{\text{nnf}})}{\vdots} \quad \frac{\overline{\neg\psi} \vdash \psi}{\overline{\neg\psi} \vdash \varphi \vee \psi} (\vee_{id})}{\overline{\neg(\varphi \wedge \overline{\psi})}, \overline{\neg\psi} \vdash \varphi \vee \psi} (\vee)}{\overline{\neg(\varphi \wedge \overline{\psi})} \vdash \varphi \vee \psi} (\vee_e)
\end{array}$$

Cas de $\varphi \wedge \psi$: alors $\text{nnf}(\varphi \wedge \psi) = \text{nnf}(\varphi) \wedge \text{nnf}(\psi)$ et $\overline{\varphi \wedge \psi} = \overline{\varphi} \vee \overline{\psi}$. On a alors la dérivation de (nnf) suivante

$$\begin{array}{c}
\text{h.i. (nnf)} \quad \text{h.i. (nnf)} \\
\vdots \quad \quad \quad \vdots \\
\frac{\varphi \vdash \text{nnf}(\varphi)}{\varphi, \psi \vdash \text{nnf}(\varphi)} (\vee) \quad \frac{\psi \vdash \text{nnf}(\psi)}{\varphi, \psi \vdash \text{nnf}(\psi)} (\vee) \\
\frac{\varphi \wedge \psi \vdash \text{nnf}(\varphi)}{\varphi \wedge \psi \vdash \text{nnf}(\varphi)} (\wedge_g) \quad \frac{\varphi \wedge \psi \vdash \text{nnf}(\psi)}{\varphi \wedge \psi \vdash \text{nnf}(\psi)} (\wedge_g) \\
\hline
\varphi \wedge \psi \vdash \text{nnf}(\varphi) \wedge \text{nnf}(\psi) \quad (\wedge_i)
\end{array}$$

Pour finir, pour $(\overline{\text{nnf}})$, on a la dérivation suivante

$$\begin{array}{c}
\text{h.i. } (\overline{\text{nnf}}) \quad \text{exemple 9.6} \quad \text{h.i. } (\overline{\text{nnf}}) \quad \text{exemple 9.6} \\
\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \\
\frac{\overline{\neg\varphi} \vdash \varphi}{\overline{\neg(\varphi \vee \overline{\psi})}, \overline{\neg\varphi} \vdash \varphi} (\vee) \quad \frac{\overline{\neg(\varphi \vee \overline{\psi})} \vdash \overline{\neg\varphi} \wedge \overline{\neg\psi}}{\overline{\neg(\varphi \vee \overline{\psi})} \vdash \overline{\neg\varphi}} (\wedge_{eg}) \quad \frac{\overline{\neg\psi} \vdash \psi}{\overline{\neg(\varphi \vee \overline{\psi})}, \overline{\neg\psi} \vdash \psi} (\vee) \quad \frac{\overline{\neg(\varphi \vee \overline{\psi})} \vdash \overline{\neg\varphi} \wedge \overline{\neg\psi}}{\overline{\neg(\varphi \vee \overline{\psi})} \vdash \overline{\neg\psi}} (\wedge_{eg}) \\
\hline
\overline{\neg(\varphi \vee \overline{\psi})} \vdash \varphi \quad \overline{\neg(\varphi \vee \overline{\psi})} \vdash \psi \\
\hline
\overline{\neg(\varphi \vee \overline{\psi})} \vdash \varphi \wedge \psi \quad (\wedge_i)
\end{array}$$

Cela conclut la démonstration. \square

11.4.4. *Complétude propositionnelle de la déduction naturelle.* Rappelons ici l'énoncé du théorème de complétude propositionnelle de la déduction naturelle.

Théorème 9.11 (complétude propositionnelle de la déduction naturelle). *Soit Γ un ensemble fini de formules propositionnelles et φ une formule propositionnelle. Si $\Gamma \models \varphi$, alors $\Gamma \vdash_{\text{NK}_0} \varphi$.*

Démonstration. Soit Γ un ensemble fini de formules propositionnelles et φ une formule propositionnelle.

Si $\Gamma \models \varphi$ alors par le lemme 6.2 de déduction, $\models \varphi \vee \bigvee_{\psi \in \Gamma} \neg\psi$. Comme la mise sous forme normale négative préserve la sémantique des formules, on a aussi $\models \text{nnf}(\varphi) \vee \bigvee_{\psi \in \Gamma} \overline{\psi}$. On peut mettre cette disjonction sous la forme d'un séquent valide $\vdash \text{nnf}(\varphi), \overline{\Gamma}$, et par le théorème 11.14 de complétude propositionnelle du calcul des séquents, ce séquent est prouvable en calcul des séquents propositionnel : $\vdash_{\text{LK}_0} \text{nnf}(\varphi), \overline{\Gamma}$. Comme $\overline{\text{nnf}(\varphi)} = \overline{\varphi}$ et $\overline{\overline{\psi}} = \text{nnf}(\psi)$, par le théorème 11.16, on a enfin $\overline{\varphi}, \text{nnf}(\Gamma) \vdash_{\text{NK}_0} \perp$ en déduction naturelle, où on a noté $\text{nnf}(\Gamma) \stackrel{\text{def}}{=} \{\text{nnf}(\psi) \mid \psi \in \Gamma\}$.

Grâce à l'admissibilité de la forme normale négative montrée dans la proposition 11.17, on a la dérivation en déduction naturelle suivante, qui montre que $\text{nnf}(\Gamma) \vdash_{\text{NK}_0} \varphi$.

$$\frac{\frac{\overline{\neg\varphi} \vdash \varphi}{\text{nnf}(\Gamma), \overline{\neg\varphi} \vdash \varphi} (\vee) \quad \frac{\overline{\varphi}, \text{nnf}(\Gamma) \vdash \perp}{\text{nnf}(\Gamma) \vdash \overline{\neg\varphi}} (\perp_c)}{\text{nnf}(\Gamma) \vdash \varphi} (\text{cut})$$

\clubsuit Si on inspecte la dérivation trouvée pour $\Gamma \vdash_{\text{NK}_0} \varphi$ dans cette démonstration de la complétude propositionnelle de la déduction naturelle, on peut observer que l'on n'a pas besoin des deux règles (\rightarrow_i) et (\rightarrow_e) du symbole d'implication pour peu que l'on se dote de la règle de coupure (cut). Ces règles sont cependant nécessaires si l'on souhaite travailler sur une syntaxe étendue avec le symbole d'implication ; la démonstration du théorème 9.11 peut être étendue pour cela.

On va montrer par récurrence sur $|\Gamma|$ le nombre de formules dans Γ que pour tout ensemble fini de formules Γ , tout ensemble Δ et toute formule φ ,

$$\text{si } \text{nnf}(\Gamma), \Delta \vdash_{\text{NK}_0} \varphi \text{ alors } \Gamma, \Delta \vdash_{\text{NK}_0} \varphi. \quad (11.1)$$

En effet, pour le cas de base où $\Gamma = \emptyset$, il n'y a rien à montrer. Pour l'étape de récurrence, si $\Gamma = \Gamma' \cup \{\psi\}$ avec $\psi \notin \Gamma'$, on a $\text{nnf}(\Gamma'), \text{nnf}(\psi), \Delta \vdash_{\text{NK}_0} \varphi$ et donc $\Gamma', \text{nnf}(\psi), \Delta \vdash_{\text{NK}_0} \varphi$ par hypothèse de récurrence appliquée à Γ' . On vérifie alors que (11.1) est encore vraie grâce à la dérivation suivante en déduction naturelle, qui utilise à nouveau l'admissibilité de la forme normale négative vue dans la proposition 11.17.

$$\frac{\frac{\Gamma', \Delta, \text{nnf}(\psi) \vdash \varphi}{\Gamma', \psi, \Delta, \text{nnf}(\psi) \vdash \varphi} \text{ (w)} \quad \frac{\overline{\psi \vdash \text{nnf}(\psi)} \text{ (nnf)}}{\Gamma', \psi, \Delta \vdash \text{nnf}(\psi)} \text{ (w)}}{\Gamma', \psi, \Delta \vdash \varphi} \text{ (cut)}$$

On peut maintenant conclure : comme $\text{nnf}(\Gamma) \vdash_{\text{NK}_0} \varphi$, en appliquant l'équation (11.1), on a aussi $\Gamma \vdash_{\text{NK}_0} \varphi$. \square

Références

Quelques ouvrages sur la logique.

- BÖRGER, Egon, Erich GRÄDEL et Yuri Sh. GUREVICH (1997). *The Classical Decision Problem. Perspectives in Mathematical Logic*. Springer-Verlag.
- CHANG, Chen C. et Howard J. KEISLER (1990). *Model Theory*. 3^e édition. Studies in Logic and the Foundations of Mathematics 73. North Holland. DOI : 10 . 1016/S0049-237X(08)70077-6.
- CHANG, Chin-Liang et Richard Char-Tung LEE (1973). *Symbolic Logic and Mechanical Theorem Proving*. Computer Science and Applied Mathematics. Academic Press. DOI : 10 . 1016/B978-0-08-091728-3 . 50001-1.
- CONCHON, Sylvain et Laurent SIMON (2018). « Satisfaisabilité propositionnelle (SAT) et modulo théories (SMT) ». Dans : *Informatique Mathématique. Une photographie en 2018*. Sous la dir. d'Emmanuel JEANDEL et Laurent VIGNERON. CNRS Éditions. Chap. 2, pp. 43-86. URL : <https://ejcim2018.sciencesconf.org/data/pages/ejcim2018-2.pdf>.
- CORI, René et Daniel LASCAR (2003). *Logique mathématique*. 2^e édition. Volume II. Fonctions récursives, théorème de GÖDEL, théorie des ensembles, théorie des modèles. Dunod.
- DAVID, René, Karim NOUR et Christophe RAFFALLI (2003). *Introduction à la logique*. 2^e édition. Dunod.
- DUPARC, Jacques (2015). *La logique pas à pas*. Presses polytechniques et universitaires romandes.
- EBBINGHAUS, Heinz-Dieter, Jörg FLUM et Wolfgang THOMAS (1994). *Mathematical Logic*. 2^e édition. Undergraduate Texts in Mathematics. Springer. DOI : 10 . 1007/978-1-4757-2355-7.
- GOUBAULT-LARRECQ, Jean et Ian MACKIE (1997). *Proof Theory and Automated Deduction*. Applied Logic Series 6. Kluwer Academic Publishers.
- HARRISSON, John (2009). *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press.
- JAUME, Mathieu, Matthieu JOURNAULT, Marie-Jeanne LESOT, Pascal MANOURY et Isabelle MOUNIER (2020). *Logique pour l'informatique*. Ellipses.
- KNUTH, Donald E. (2008). *The Art of Computer Programming*. Volume 4, fascicule 0 : *Introduction to Combinatorial Algorithms and Boolean Functions*. Addison-Wesley. URL : <https://cs.stanford.edu/~knuth/fasc0b.ps.gz>.

- KNUTH, Donald E. (2011). *The Art of Computer Programming*. Volume 4, fascicule 1 : *Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley. URL : <https://cs.stanford.edu/~knuth/fasc1b.ps.gz>.
- (2015). *The Art of Computer Programming*. Volume 4, fascicule 6 : *Satisfiability*. Addison-Wesley. URL : <https://cs.stanford.edu/~knuth/fasc6a.ps.gz>.
- LALEMENT, René (1990). *Logique, réduction, résolution*. Études et recherches en informatique. Masson.
- LASSAIGNE, Richard et Michel de ROUGEMONT (2004). *Logic and Complexity*. Discrete Mathematics and Theoretical Computer Science. Springer. DOI : 10.1007/978-0-85729-392-3.
- LIBKIN, Leonid (2004). *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer. DOI : 10.1007/978-3-662-07003-1. URL : <https://homepages.inf.ed.ac.uk/libkin/fmt/fmt.pdf>.
- MANCOSU, Paulo, Sergio GALVAN et Richard ZACH (2021). *An Introduction to Proof Theory. Normalization, Cut-Elimination, and Consistency Proofs*. Oxford University Press. DOI : 10.1093/oso/9780192895936.001.0001.
- TROELSTRA, Arne S. et Helmut SCHWICHTENBERG (2000). *Basic Proof Theory*. 2^e édition. Cambridge Tracts in Theoretical Computer Science 43. Cambridge University Press. DOI : 10.1017/CBO9781139168717.

Quelques textes fondateurs.

- BERGER, Robert (1966). *The Undecidability of the Domino Problem*. Memoirs of the American Mathematical Society 66. American Mathematical Society. DOI : 10.1090/memo/0066.
- BLAKE, Archie (1937). « Canonical expressions in Boolean algebra ». Thèse de doct. Department of Mathematics, University of Chicago.
- CHURCH, Alonzo (1936). « An unsolvable problem of elementary number theory ». Dans : *American Journal of Mathematics* 58(2), pp. 345-363. DOI : 10.2307/2371045.
- COOK, Stephen A. (1971). « The complexity of theorem proving procedures ». Dans : *Actes de STOC '71*. ACM, pp. 151-158. DOI : 10.1145/800157.805047.
- COOK, Stephen A. et Robert A. RECKHOW (1979). « The relative efficiency of propositional proof systems ». Dans : *Journal of Symbolic Logic* 44(1), pp. 36-50. DOI : 10.2307/2273702.
- DAVIS, Martin, George LOGEMANN et Donald LOVELAND (1961). « A machine program for theorem proving ». Dans : *Communications of the ACM* 5(7), pp. 394-397. DOI : 10.1145/368273.368557.
- DAVIS, Martin et Hilary PUTNAM (1960). « A computing procedure for quantification theory ». Dans : *Journal of the ACM* 7(3), pp. 201-215. DOI : 10.1145/321033.321034.
- DE BRUIJN, Nicolaas G. et Paul ERDŐS (1951). « A colour problem for infinite graphs and a problem in the theory of relations ». Dans : *Indagationes Mathematicae (Proceedings)* 54, pp. 371-373. DOI : 10.1016/S1385-7258(51)50053-7.
- KARP, Richard M. (1972). « Reducibility among combinatorial problems ». Dans : *Actes du symposium « Complexity of Computer Computations »*. Springer, pp. 85-103. DOI : 10.1007/978-1-4684-2001-2_9.

- LEVIN, Leonid (1973). « Universal sequential search problems ». Dans : *Problems of Information Transmission* 9(3), pp. 115-116. URL : <http://mi.mathnet.ru/eng/ppi914>.
- МАТИАСЕВИЧ, Iouri V. (1970). « Les ensembles énumérables sont diophantiens ». Dans : *Comptes rendus de l'Académie des sciences de l'URSS* 191(2). En russe original : Матиясевиц, Юрий В. (1970). « Диофантовость перечислимых множеств ». Dans : *Доклады Академии наук СССР* 191(2), pp. 279-282. URL : <http://mi.mathnet.ru/dan35274>.
- QUINE, Willard Van Orman (1952). « The problem of simplifying truth functions ». Dans : *The American Mathematical Monthly* 59(8), pp. 521-531. DOI : 10.2307/2308219.
- ROBINSON, J. Alan (1965). « A machine-oriented logic based on the resolution principle ». Dans : *Journal of the ACM* 12(1), pp. 23-41. DOI : 10.1145/321250.321253.
- STOCKMEYER, Larry J. (1974). « The Complexity of Decision Problems in Automata Theory and Logic ». Thèse de doct. Massachusetts Institute of Technology. Dept. of Electrical Engineering. URL : <http://hdl.handle.net/1721.1/15540>.
- ТРАХТЕНБРОТ, Boris A. (1950). « L'impossibilité d'un algorithme pour le problème de décision sur les classes finies ». Dans : *Comptes-rendus de l'Académie des sciences de l'URSS* 70(4). En russe original : Трахтенброт, Б. А. (1950). « Невозможность алгоритма для проблемы разрешимости на конечных классах ». Dans : *Доклады Академии наук СССР* 70(4), pp. 569-572.
- TSEITIN, Grigori S. (1966). « On the complexity of derivation in propositional calculus ». Dans : *Actes du Séminaire de Leningrad sur la logique mathématique*. URL : <http://www.decision-procedures.org/handouts/Tseitin70.pdf>.
- TURING, Alan M. (1937). « On computable numbers, with an application to the Entscheidungsproblem ». Dans : *Proceedings of the London Mathematical Society*. 2^e sér. 42(1), pp. 230-265. DOI : 10.1112/plms/s2-42.1.230. Correction en 1938 dans : *Proceedings of the London Mathematical Society* 2^e sér. 43(1), pp. 544-546. DOI : 10.1112/plms/s2-43.6.544.

Autres ouvrages.

- ABITEBOUL, Serge, Richard HULL et Victor VIANU (1995). *Foundations of Databases*. Addison Wesley. URL : <http://webdam.inria.fr/Alice/pdfs/all.pdf>.
- ARORA, Sanjeev et Boaz BARAK (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press. DOI : 10.1017/CBO9780511804090.
- BAADER, Franz et Tobias NIPKOW (1998). *Term Rewriting and All That*. Cambridge University Press. DOI : 10.1017/CBO9781139172752.
- BEAUQUIER, Danièle, Jean BERSTEL et Chrétienne PHILIPPE (1992). *Éléments d'algorithmique*. Masson. URL : <http://www-igm.univ-mlv.fr/~berstel/Elements/Elements.html>.
- CARTON, Olivier (2008). *Langages formels, calculabilité et complexité*. Vuibert.
- CHINAL, Jean (1967). *Techniques booléennes et calculateurs arithmétiques*. Techniques de l'automatisme. Dunod. DOI : 10.1007/978-3-642-86187-1.
- CORMEN, Thomas H., Charles E. LEISERSON, Ronald L. RIVEST et Clifford STEIN (2009). *Introduction to Algorithms*. 3^e édition. MIT Press.

- FLUM, Jörg et Martin GROHE (2006). *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer. DOI : 10.1007/3-540-29953-X.
- GAREY, Michael R. et David S. JOHNSON (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. W. H. Freeman et Co.
- PAPADIMITRIOU, Christos (1993). *Computational Complexity*. Addison-Wesley.
- PERIFEL, Sylvain (2014). *Complexité algorithmique*. Ellipses. URL : <https://www.irif.fr/~sperifel/complexite.pdf>.
- WEGENER, Ingo (2000). *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM Monographs in Discrete Mathematics and Applications. SIAM. DOI : 10.1137/1.9780898719789.

Autres références.

- BREITBART, Yuri, Harry B. HUNT III et Daniel J. ROSENKRANTZ (1995). « On the size of binary decision diagrams representing Boolean functions ». Dans : *Theoretical Computer Science* 145(1-2), pp. 45-69. DOI : 10.1016/0304-3975(94)00181-H.
- DYCKHOFF, Roy (1992). « Contraction-free sequent calculi for intuitionistic logic ». Dans : *Journal of Symbolic Logic* 57(3), pp. 795-807. DOI : 10.2307/2275431.
- GUREVICH, Yuri Sh. et I. O. KORYAKOV (1972). « Remarks on Berger's paper on the domino problem ». Dans : *Siberian Mathematics Journal* 13, pp. 319-321. DOI : 10.1007/BF00971620.
- HUET, Gérard (1976). « Résolution d'équations dans les langages d'ordre 1, 2, ..., ω ». Thèse de doctorat d'état. Université Paris VII.
- MANCINELLI, Fabio, Jaap BOENDER, Roberto DI COSMO, Jérôme VOULLON, Berke DURAK, Xavier LEROY et Ralf TREINEN (2006). « Managing the complexity of large free and open source package-based software distributions ». Dans : *Actes de ASE '06*. IEEE, pp. 199-208. DOI : 10.1109/ASE.2006.49. URL : <https://hal.archives-ouvertes.fr/hal-00149566/>.
- MARTELLI, Alberto et Ugo MONTANARI (1982). « An efficient unification algorithm ». Dans : *ACM Transactions on Programming Languages and Systems* 4(2), pp. 258-282. DOI : 10.1145/357162.357169.
- PATERSON, Michael S. et Mark N. WEGMAN (1978). « Linear unification ». Dans : *Journal of Computer and System Sciences* 16(2), pp. 158-167. DOI : 10.1016/0022-0000(78)90043-0.
- TARJAN, Robert E. et Jan van LEEUWEN (1984). « Worst-case analysis of set union algorithms ». Dans : *Journal of the ACM* 31(2), pp. 245-281. DOI : 10.1145/62.2160.
- UMANS, Christopher (2001). « The minimum equivalent DNF problem and shortest implicants ». Dans : *Journal of Computer and System Sciences* 63(4), pp. 597-611. DOI : 10.1006/jcss.2001.1775.
- ZHANG, Lintao, C.F. MADIGAN, M.H. MOSKEWICZ et S. MALIK (2001). « Efficient conflict driven learning in a Boolean satisfiability solver ». Dans : *Actes de ICCAD '01*, pp. 279-285. DOI : 10.1109/ICCAD.2001.968634.