Logique propositionnelle

Sylvain Schmitz

LOGIQUE PROPOSITIONNELLE
NOTES DE RÉVISION POUR L'AGRÉGATION

SYLVAIN SCHMITZ

ENS Paris-Saclay, France

Résumé. Ces notes de révision sont consacrées à la leçon **916** « Formules du calcul propositionnel : représentation, formes normales, satisfiabilité. » de l'agrégation de mathématiques, option D, session 2019.

Contenu des notes

Le programme officiel 2019 pour cette leçon est [1] :

> *Calcul propositionnel : syntaxe et sémantique. Tables de vérité, tautologies, formes normales, forme clausale. Théorème de complétude du calcul propositionnel.*

Le rapport du jury 2018 pour cette leçon en précise le contenu attendu [2] :

> *Le jury attend des candidats qu'ils abordent les questions de la complexité de la satisfiabilité. Pour autant, les applications ne sauraient se réduire à la réduction de problèmes* NP*-complets à* SAT*. Une partie significative du plan doit être consacrée à la représentation des formules et à leurs formes normales.*

Ces notes de révision couvrent globalement le programme de l'agrégation, à l'exception des tables de vérité que les candidat·e·s trouveront dans n'importe quel ouvrage sur la question.

Le point de vue de ces notes de révision est avant tout *algorithmique*. La logique propositionnelle est en effet un formalisme dans lequel de nombreux problèmes informatiques peuvent s'exprimer comme des problèmes de *satisfiabilité*. Bien que ce dernier problème soit NP-complet, les solveurs SAT sont redoutablement efficaces sur des instances qui apparaissent en pratique : il n'est pas inhabituel qu'en pratique on obtienne de meilleures performances avec un solveur SAT fortement optimisé qu'avec un algorithme déterministe en temps $O(n^d)$ quand le degré d atteint trois ou quatre. Enfin, la restriction à HornSAT fournit aussi un algorithme très efficace pour quantité de problèmes P-complets.

Partie 1. Logique classique propositionnelle 3
1. Syntaxe 3
2. Sémantique 3
3. Substitutions propositionnelles 4
4. Compacité 5
4.1. Compacité par le théorème de Tychonoff 5
4.2. Compacité par arbres sémantiques 6

1. http://media.devenirenseignant.gouv.fr/file/agregation_externe/13/1/p2019_agreg_ext_maths_929131.pdf#section.16
2. http://agreg.org/Rapports/rapport2018.pdf#subsection.5.4.3

Partie 2. Formes normales 8
5. Forme normale négative 8
6. Forme clausale 8
6.1. Forme clausale équivalente 8
6.2. Forme clausale équi-satisfiable 9
6.3. Forme normale disjonctive 10
7. Complétude fonctionnelle 10
7.1. Exemples de bases complètes 11
7.2. Exemple de base incomplète 11
8. Diagrammes binaires de décision 12
8.1. Formules de Shannon ordonnées 12
8.2. Formules de Shannon complètes 13
8.3. Formules de Shannon réduites 14

Partie 3. Satisfiabilité 15
9. Calcul des séquents propositionnel 15
9.1. Recherche de preuve 16
9.2. Correction et complétude 18
10. Résolution propositionnelle 19
10.1. Complétude réfutationnelle 20
10.2. Complétude propositionnelle 22
11. Algorithme de Davis, Putnam, Logemann et Loveland 23
11.1. Recherche de modèle par simplification 23
11.2. Branches de succès et algorithme DPLL 24
12. Clauses de Horn 26
12.1. Modèle minimal 26
12.2. Complexité 28
12.3. Application à la preuve par saturation en résolution 31
13. 2SAT 31

Annexe. 35
Annexe A. Recherche de réfutation en résolution propositionnelle 35

Références 37

L'objectif de ces notes de révision est de fournir une référence avec des notations unifiées et des pointeurs vers différents ouvrages susceptibles d'être utilisés lors de la préparation au concours de l'agrégation. Ces notes ne sont pas un substitut à l'étude approfondie d'ouvrages et d'articles sur le sujet, référencés par le symbole « 🔖 » dans les notes en marge ; les deux principaux ouvrages sur lesquels je m'appuie sont :

René David, Karim Nour et Christophe Raffalli, 2003. *Introduction à la logique.* Dunod, Paris, 2e édition.

Jean Goubault-Larrecq et Ian Mackie, 1997. *Proof Theory and Automated Deduction*, volume 6 de *Applied Logic Series.* Kluwer Academic Publishers, Amsterdam.

Une partie de ces notes est inspirée des notes de cours du « MOOC » *Introduction à la logique informatique* par David Baelde, Hubert Comon et Étienne Lozes [3].

3. Voir les pages https://www.fun-mooc.fr/courses/ENSCachan/20004S02/session02/about et https://www.fun-mooc.fr/courses/ENSCachan/20009/session01/about

Partie 1. Logique classique propositionnelle

Commençons, afin de fixer les notations employées dans ces notes, par rappeler la syntaxe et la sémantique de la logique propositionnelle (aussi appelée « calcul des propositions »).

1. Syntaxe

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

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

où $P \in \mathcal{P}_0$. Un *littéral* est une proposition P ou sa négation $\neg P$.

On peut ajouter d'autres symboles booléens à cette syntaxe minimale (ou choisir une autre syntaxe fonctionnellement complète, *c.f.* section 7). Alternativement, ces symboles peuvent être définis dans la logique ; par exemple :

$$\varphi \wedge \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \vee \neg\psi)\ , \qquad \varphi \Rightarrow \psi \stackrel{\text{def}}{=} \neg\varphi \vee \psi\ , \qquad \top \stackrel{\text{def}}{=} P \vee \neg P\ , \qquad \bot \stackrel{\text{def}}{=} \neg\top\ ,$$

où P est n'importe quelle proposition tirée de $\mathcal{P}_0$ (qui est non vide puisque supposé infini).

L'ensemble $\mathcal{P}_0(\varphi)$ des *propositions* d'une formule propositionnelle φ est défini inductivement par

$$\mathcal{P}_0(P) \stackrel{\text{def}}{=} \{P\}\ , \qquad \mathcal{P}_0(\neg\varphi) \stackrel{\text{def}}{=} \mathcal{P}_0(\varphi)\ , \qquad \mathcal{P}_0(\varphi \vee \psi) \stackrel{\text{def}}{=} \mathcal{P}_0(\varphi) \cup \mathcal{P}_0(\psi)\ .$$

Remarquons enfin qu'en informatique, les formules sont couramment représentées par des graphes acycliques dirigés, où les sous-graphes isomorphes sont identifiés, plutôt que par des arbres. On parlera dans ces notes de *circuit* associé à la formule ; dans ce contexte, les sommets du graphe sont appelés des *portes* et comme le graphe est acyclique on peut raisonner par induction structurelle sur les portes comme on le ferait sur les sous-arbres dans une représentation arborescente. Par exemple, la figure 1 décrit ces deux représentations mémoire de la loi de Pierce $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$.

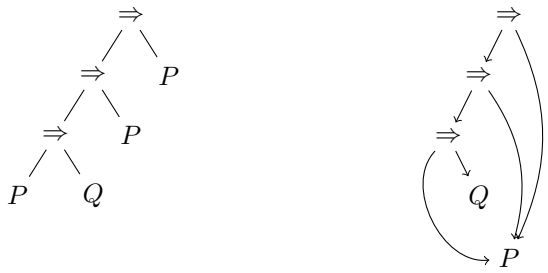

Figure 1. Représentation mémoire de la loi de Pierce sous forme d'arbre et sous forme de circuit.

La différence entre ces deux représentations n'est pas anodine : l'espace mémoire nécessaire pour représenter une formule sous forme de circuit peut être exponentiellement plus petit que celui pour une formule équivalente sous forme d'arbre.

2. Sémantique

Valeurs de vérité. On note $\mathbb{B} \stackrel{\text{def}}{=} \{\bot, \top\}$ pour l'ensemble des *valeurs de vérité* (aussi appelé « algèbre de Boole »), muni des opérations $\neg\colon \mathbb{B} \to \mathbb{B}$ et $\wedge, \vee\colon \mathbb{B}^2 \to \mathbb{B}$ définies par $\neg\top = \bot \vee \bot = \bot \wedge \bot = \top \wedge \bot = \bot \wedge \top = \bot$ et $\neg\bot = \top \vee \top = \top \vee \bot = \bot \vee \top = \top \wedge \top = \top$.

[David et al., 2003, sec. 1.2.6], [Goubault-Larrecq et Mackie, 1997, sec. 2.1]

⚠ *Utiliser une syntaxe de mots à la place d'une syntaxe de termes serait un* hors-sujet*, même si vous aviez envie de montrer votre maîtrise de la leçon* ***923****.*

💬 *On peut cependant traduire un circuit en une formule* équi-satisfiable *en temps linéaire, en introduisant des propositions supplémentaires, voir la section 6.2.*

Interprétations. Une *interprétation* $I \in \mathbb{B}^{\mathcal{P}_0}$ (aussi appelée une « valuation propositionnelle ») est constituée d'une valeur de vérité $P^I \in \mathbb{B}$ pour chaque proposition $P \in \mathcal{P}_0$. On verra aussi une interprétation comme un sous-ensemble $I \stackrel{\text{def}}{=} \{P \in \mathcal{P}_0 \mid P^I = \top\}$ de propositions dans $2^{\mathcal{P}_0}$; les deux points de vue sont bien sûr équivalents. Si I est une interprétation et P est une proposition, on écrit $I[\top/P]$ (resp. $I[\bot/P]$) pour l'interprétation qui associe $\top$ à P (resp. $\bot$) et Q^I à Q pour tout $Q \neq P$.

On considérera parfois des interprétations *partielles* $I : \mathbb{B}^{\mathcal{P}_0} \rightharpoonup \mathbb{B}$ définies sur un ensemble $\mathrm{dom}(I) \subseteq \mathcal{P}_0$. Pour deux interprétations partielles I et I', on dit que I' *étend* I ou que I est la *restriction* de I' à $\mathrm{dom}(I)$, et on écrit $I \sqsubseteq I'$, si $\mathrm{dom}(I) \subseteq \mathrm{dom}(I)'$ et pour toute proposition $P \in \mathrm{dom}(I)$, $P^I = P^{I'}$.

💬 *C'est une sémantique dénotationnelle au sens de la leçon **930**.*

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

$$[\![P]\!]^I \stackrel{\text{def}}{=} P^I \ , \qquad [\![\neg\varphi]\!]^I \stackrel{\text{def}}{=} \neg[\![\varphi]\!]^I \ , \qquad [\![\varphi \vee \psi]\!]^I \stackrel{\text{def}}{=} [\![\varphi]\!]^I \vee [\![\psi]\!]^I \ .$$

On dit que I *satisfait* φ (ou que I est un « modèle » de φ), noté $I \vDash \varphi$, si $[\![\varphi]\!]^I = \top$; cette écriture peut être définie de manière équivalente par

$$\begin{aligned} &I \vDash P && \text{si } P \in I \ , \\ &I \vDash \neg\varphi && \text{si } I \nvDash \varphi \ , \\ &I \vDash \varphi \vee \psi && \text{si } I \vDash \varphi \text{ ou } I \vDash \psi \ . \end{aligned}$$

On définit aussi $\mathrm{Sat}(\varphi) \stackrel{\text{def}}{=} \{I \in \mathbb{B}^{\mathcal{P}_0} \mid I \vDash \varphi\}$ l'ensemble des interprétations qui satisfont φ. On peut observer que la satisfaction de φ ne dépend que des propositions de $\mathcal{P}_0(\varphi)$.

Propriété 2.1. *Pour toute formule propositionnelle φ et interprétations I et I', si $I \cap \mathcal{P}_0(\varphi) = I' \cap \mathcal{P}_0(\varphi)$, alors $I \vDash \varphi$ si et seulement si $I' \vDash \varphi$.*

Une formule φ est *satisfiable* s'il existe un modèle de φ. Elle est *valide*, noté $\vDash \varphi$, si pour toute interprétation I, $I \vDash \varphi$.

💬 *Cette formule n'est pas valide en logique intuitionniste.*

Exemple 2.2 (loi de Pierce)**.** La formule $\varphi \stackrel{\text{def}}{=} ((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$ est une variante du tiers exclu, et est valide en logique classique propositionnelle. En effet, soit I une interprétation quelconque. Si $I \vDash P$, alors $I \vDash \varphi$. Sinon, $I \vDash P \Rightarrow Q$ donc $I \nvDash (P \Rightarrow Q) \Rightarrow P$ et donc $I \vDash \varphi$.

Pour un ensemble de formules propositionnelles S et une interprétation I, on écrit $I \vDash S$ si $I \vDash \psi$ pour tout $\psi \in S$. Si S est un ensemble de formules et φ est une formule, on écrit $S \vDash \varphi$ si pour toute interprétation I telle que $I \vDash S$ on a $I \vDash \varphi$. Un ensemble S est *insatisfiable* s'il n'existe pas d'interprétation I telle que $I \vDash S$; cela est équivalent à $S \vDash \bot$.

Exemple 2.3 (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 $I \vDash P$. Si $I \vDash R$, alors $I \nvDash \neg P \vee \neg R$; sinon si $I \vDash Q$ alors $I \nvDash \neg P \vee Q \vee \neg R$ et sinon $I \nvDash Q \vee R$. Supposons maintenant $I \nvDash P$. Si $I \vDash Q$, alors $I \nvDash P \vee \neg Q$; sinon si $I \vDash R$ alors $I \nvDash P \vee Q \vee \neg R$ et sinon $I \nvDash Q \vee R$.

3. Substitutions propositionnelles

🔖 *[Goubault-Larrecq et Mackie, 1997, def. 2.4], [David et al., 2003, def. 4.6.4]*

Une *substitution propositionnelle* est une fonction σ de $\mathcal{P}_0$ dans les formules propositionnelles telle que $\mathrm{dom}(\sigma) \stackrel{\text{def}}{=} \{P \in \mathcal{P}_0 \mid \sigma(P) \neq P\}$ soit fini. On écrit $[\varphi_1/P_1, \ldots, \varphi_n/P_n]$ pour la substitution de domaine $\{P_1, \ldots, P_n\}$ où les P_i sont distinctes qui associe φ_i à P_i. Toute substitution

se relève en une fonction des formules propositionnelles dans les formules propositionnelles par homomorphisme :

$$P\sigma \stackrel{\text{def}}{=} \sigma(P)\ , \qquad (\neg\varphi)\sigma \stackrel{\text{def}}{=} \neg(\varphi\sigma)\ , \qquad (\varphi \vee \psi)\sigma \stackrel{\text{def}}{=} (\varphi\sigma) \vee (\psi\sigma)\ .$$

Pour une interprétation I et une substitution propositionnelle σ, on définit l'interprétation $I\sigma$ comme associant $[\![\sigma(P)]\!]^I$ à chaque proposition P.

Lemme 3.1 (substitution propositionnelle)**.** *Pour toute formule propositionnelle φ, toute substitution propositionnelle σ et toute interprétation I, $[\![\varphi\sigma]\!]^I = [\![\varphi]\!]^{I\sigma}$.*

[Goubault-Larrecq et Mackie, 1997, thm. 2.10]

Démonstration. Par induction structurelle sur φ. Pour le cas de base d'une proposition P,

$$[\![P\sigma]\!]^I = [\![\sigma(P)]\!]^I = [\![P]\!]^{I\sigma}$$

par définition. Pour l'étape d'induction pour $\neg\varphi$,

$$[\![(\neg\varphi)\sigma]\!]^I = [\![\neg(\varphi\sigma)]\!]^I = \neg[\![\varphi\sigma]\!]^I \stackrel{\text{h.i.}}{=} \neg[\![\varphi]\!]^{I\sigma} = [\![\neg\varphi]\!]^{I\sigma}\ .$$

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

$$[\![(\varphi \vee \psi)\sigma]\!]^I = [\![(\varphi\sigma) \vee (\psi\sigma)]\!]^I = [\![\varphi\sigma]\!]^I \vee [\![\psi\sigma]\!]^I \stackrel{\text{h.i.}}{=} [\![\varphi]\!]^{I\sigma} \vee [\![\psi]\!]^{I\sigma} = [\![\varphi \vee \psi]\!]^{I\sigma}\ . \qquad \square$$

4. 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, $S \vDash \bot$ si et seulement si $\exists F \subseteq_{\text{fin}} S$ tel que $F \vDash \bot$.

Théorème 4.1 (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 4.1 ;
- par la construction d'arbres sémantiques, voir la section 4.2 ;
- comme une conséquence de la correction et complétude de systèmes de preuves : $S \vDash \bot$ implique $S \vdash \bot$, mais comme les preuves sont finies, il existe un sous-ensemble fini F de S tel que $F \vdash \bot$, et par correction $F \vDash \bot$; c'est ce qui sera utilisé en section 10.1.2.

Voici enfin une formulation équivalente du théorème.

Corollaire 4.2. *Soit S un ensemble de formules propositionnelles et φ une formule propositionnelle. Si $S \vDash \varphi$, alors il existe un sous-ensemble fini F de S tel que $F \vDash \varphi$.*

Démonstration. Si $S \vDash \varphi$, 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 $I \nvDash \varphi$ alors $I \nvDash F$, sans quoi on aurait $I \vDash F'$ qui contredirait son insatisfiabilité. $\square$

4.1. Compacité par le théorème de Tychonoff.

Cette preuve est très simple mais nécessite d'introduire quelques notions de base en topologie. Ces notions sont étudiées dans la leçon **203** dans le cas des espaces métriques, mais elles s'appliquent plus généralement aux espaces topologiques. 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.

4.1.1. *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 $T = \bigcap_\emptyset$ et $\emptyset = \bigcup_\emptyset$ sont ouverts. Les ensembles *fermés* sont définis comme les compléments $T \setminus O$ d'ouverts ; 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 T_j. 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 T_j.

Théorème 4.3 (Tychonoff)**.** *Soit $(T_j)_{j\in J}$ une famille d'espaces topologiques compacts. Alors $\prod_{j\in J} T_j$ est compact.*

[Goubault-Larrecq et Mackie, 1997, thm. 2.25]

Démonstration du théorème de compacité. L'ensemble $\mathbb{B} = \{\bot, \top\}$ 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 $\mathrm{Sat}(\varphi)$ 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, $\mathrm{Sat}(P)$ est le produit $\{\top\} \times \prod_{P'\neq P} \mathbb{B}$ associant $\top$ à P et $\mathbb{B} = \{\bot, \top\}$ à toutes les propositions $P' \neq P$, qui est bien un ouvert fermé. Pour l'étape d'induction, $\mathrm{Sat}(\neg\varphi) = \mathbb{B}^{\mathcal{P}_0} \setminus \mathrm{Sat}(\varphi)$ est bien un ouvert fermé et $\mathrm{Sat}(\varphi \vee \psi) = \mathrm{Sat}(\varphi) \cup \mathrm{Sat}(\psi)$ 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 $\mathbb{B}^{\mathcal{P}_0} = \bigcup_{\varphi\in S} \mathrm{Sat}(\neg\varphi)$. Par compacité, $\mathbb{B}^{\mathcal{P}_0}$ a un recouvrement fini $\bigcup_{\varphi\in F} \mathrm{Sat}(\neg\varphi)$; cet ensemble $F \subseteq S$ est fini et insatisfiable. □

[Chang et Lee, 1973, sec. 4.4]

4.2. **Compacité par arbres sémantiques.** Soit S un ensemble de formules et $\mathcal{P}_0(S)$ l'ensemble des propositions qui apparaissent dans S. Un *arbre sémantique* pour S est un arbre binaire, généralement infini si $\mathcal{P}_0(S)$ est infini, dont les nœuds sont étiquetés par des propositions de $\mathcal{P}_0(S)$ et tel que, pour toute branche de l'arbre, chaque proposition de $\mathcal{P}_0(S)$ étiquette exactement un nœud de la branche.

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

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 $[\bot/P]$ et le fils droit à I étendue par $[\top/P]$. Une branche de l'arbre sémantique est donc associée à une interprétation de domaine $\mathcal{P}_0(S)$ entier. Pour une interprétation partielle I et une formule φ, on écrit $I \vDash \varphi$ s'il existe une interprétation totale I' qui étend I (c'est-à-dire avec $P^I = P^{I'}$ pour tout $P \in \mathrm{dom}(I)$)

telle que $I' \vDash \varphi$. 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 $I \nvDash \varphi$.

Supposons S insatisfiable. Pour toute interprétation I, il existe une formule $\varphi \in S$ telle que $I \nvDash \varphi$. 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 $I \nvDash \varphi$ où I est l'interprétation partielle du nœud. Par le lemme de Kőnig, un arbre sémantique fermé est nécessairement fini. La figure 2 montre un arbre sémantique fermé pour l'exemple 2.3.

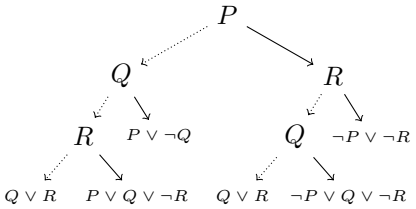

FIGURE 2. Arbre sémantique fermé pour l'ensemble de formules de l'exemple 2.3.

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 $I \nvDash \varphi$. Donc F est insatisfiable. □

Partie 2. Formes normales

Les formules propositionnelles peuvent être mises sous différentes formes normales préservant la sémantique (formules dites *équivalentes*) ou préservant la satisfiabilité (formules dites *équi-satisfiables*).

5. Forme normale négative

[David et al., 2003, sec. 2.6], [Goubault-Larrecq et Mackie, 1997, def. 2.38]

Une formule du premier ordre est en *forme normale négative* si elle respecte la syntaxe abstraite

$$\ell ::= P \mid \neg P \qquad \text{(littéraux)}$$
$$\varphi ::= \ell \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \qquad \text{(formules)}$$

où P est une proposition de $\mathcal{P}_0$. En d'autres termes, les négations ne peuvent apparaître que devant des formules atomiques. La mise sous forme normale négative procède en « poussant » les négations vers les feuilles ; pour une formule φ, on notera $\mathrm{nnf}(\varphi)$ sa forme normale négative obtenue inductivement par

$$\mathrm{nnf}(P) \stackrel{\text{def}}{=} P\ , \qquad \mathrm{nnf}(\neg P) \stackrel{\text{def}}{=} \neg P\ ,$$
$$\mathrm{nnf}(\varphi \vee \psi) \stackrel{\text{def}}{=} \mathrm{nnf}(\varphi) \vee \mathrm{nnf}(\psi)\ , \qquad \mathrm{nnf}(\varphi \wedge \psi) \stackrel{\text{def}}{=} \mathrm{nnf}(\varphi) \wedge \mathrm{nnf}(\psi)\ ,$$
$$\mathrm{nnf}(\neg(\varphi \vee \psi)) \stackrel{\text{def}}{=} \mathrm{nnf}(\neg\varphi) \wedge \mathrm{nnf}(\neg\psi)\ , \qquad \mathrm{nnf}(\neg(\varphi \wedge \psi)) \stackrel{\text{def}}{=} \mathrm{nnf}(\neg\varphi) \vee \mathrm{nnf}(\neg\psi)\ ,$$
$$\mathrm{nnf}(\neg\neg\varphi) \stackrel{\text{def}}{=} \mathrm{nnf}(\varphi)\ .$$

En termes algorithmiques, cette mise sous forme normale négative se fait en temps linéaire. Elle préserve visiblement la sémantique des formules : φ et $\mathrm{nnf}(\varphi)$ sont équivalentes. On notera en général

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

pour la forme normale négative de la négation de φ.

Exemple 5.1. La loi de Pierce $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$ s'écrit $\neg(\neg(\neg P \vee Q) \vee P) \vee P$ dans notre syntaxe. Sa forme normale négative est

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

La *profondeur* $p(\varphi)$ d'une formule φ en forme normale négative est définie inductivement par

$$p(\ell) \stackrel{\text{def}}{=} 0\ , \qquad p(\varphi \vee \psi) \stackrel{\text{def}}{=} p(\varphi \wedge \psi) \stackrel{\text{def}}{=} 1 + \max\{p(\varphi), p(\psi)\}\ .$$

À noter que la profondeur d'une formule est la même, qu'elle soit représentée sous forme d'arbre ou sous forme de circuit comme dans la figure 1.

6. Forme clausale

[David et al., 2003, sec. 7.4.2]

6.1. **Forme clausale équivalente.** Soit φ une formule propositionnelle en forme normale négative. En utilisant la distributivité, on obtient une mise sous *forme normale conjonctive* $\mathrm{cnf}(\varphi)$ pour toute φ en forme normale négative :

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

Une formule sous forme normale conjonctive s'écrit donc comme $\bigwedge_{1\le i\le m} \bigvee_{1\le j\le n_i} \ell_{i,j}$ où les $\ell_{i,j}$ sont des littéraux ; les sous-formules $C_i \stackrel{\text{def}}{=} \bigvee_{1\le j\le n_i} \ell_{i,j}$ sont les *clauses* de $\mathrm{cnf}(\varphi)$. Pour une formule φ donnée, on note $\mathrm{Cl}(\varphi)$ l'ensemble des clauses de $\mathrm{cnf}(\varphi)$, qu'on appelle sa *forme clausale*. Quand les clauses sont des disjonctions d'au plus k littéraux, on dit que S est sous

forme k-*clausale* (aussi appelée « k-CNF »). On peut voir une clause comme un ensemble de littéraux, pour lequel les notations ensemblistes $\in$ et $\subseteq$ s'appliquent.

Propriété 6.1. *Soit φ une formule propositionnelle et I une interprétation. Alors $I \vDash \varphi$ si et seulement si $I \vDash \mathrm{Cl}(\mathrm{nnf}(\varphi))$.*

Exemple 6.2. La *négation* de la loi de Pierce $((P \Rightarrow Q) \Rightarrow P) \wedge \neg P$ s'écrit $((P \wedge \neg Q) \vee P) \wedge \neg P$ en forme normale négative. La mise sous forme conjonctive produit $(P \vee P) \wedge (\neg Q \vee P) \wedge \neg P$ et donc l'ensemble de clauses $\{P \vee P, \neg Q \vee P, \neg P\}$.

Exemple 6.3. La mise sous forme normale conjonctive peut avoir un coût exponentiel du fait des duplications de formules. Par exemple, la formule en forme normale disjonctive $\varphi \stackrel{\text{def}}{=} \bigvee_{1 \leq i \leq n} P_i \wedge Q_i$ est satisfaite par une interprétation I si et seulement s'il existe $1 \leq i \leq n$ tel que $\{P_i, Q_i\} \subseteq I$. La construction précédente fournit la normale conjonctive $\mathrm{cnf}(\varphi) = \bigwedge_{J \subseteq \{1,\dots,n\}} \bigvee_{1 \leq i \leq n} \ell_{J,i}$ où $\ell_{J,i} = P_i$ si $i \in J$ et $\ell_{J,i} = Q_i$ sinon.

6.2. **Forme clausale équi-satisfiable.** En général, étant donné un circuit φ sous forme normale négative, on peut construire en temps linéaire une formule équi-satisfiable représentée sous forme arborescente.

[Perifel, 2014, prop. 3-Z], [Goubault-Larrecq et Mackie, 1997, exo. 2.23], [David et al., 2003, def. 7.4.15], [Carton, 2008, p. 191]

Pour chaque porte (c'est-à-dire chaque sous-formule) φ' du circuit φ, on introduit pour cela une proposition fraîche $Q_{\varphi'} \notin \mathcal{P}_0(\varphi)$ et on définit la formule

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

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

Dans le cadre de la mise sous forme clausale, cette transformation est parfois appelée « transformation de Tseitin », qui historiquement utilisait des équivalences $Q_{\varphi'} \Leftrightarrow \psi_{\varphi'}$ au lieu d'implications.

(1) elle est sous forme *3-clausale* : c'est en effet une conjonction où les implications $Q_{\varphi'} \Rightarrow \psi_{\varphi'}$ s'écrivent comme $(\neg Q_{\varphi'} \vee Q_{\varphi_1}) \wedge (\neg Q_{\varphi'} \vee Q_{\varphi_2})$ si $\varphi' = \varphi_1 \wedge \varphi_2$ et comme $(\neg Q_{\varphi'} \vee Q_{\varphi_1} \vee Q_{\varphi_2})$ si $\varphi' = \varphi_1 \vee \varphi_2$.
(2) sa représentation arborescente est de *taille linéaire* en la taille du circuit φ : il y a une implication $Q_{\varphi'} \Rightarrow \psi_{\varphi'}$ par porte du circuit, et ces implications sont de taille bornée par une constante.

Proposition 6.4. *Le circuit φ et la formule $\psi = Q_\varphi \wedge \bigwedge_{\varphi' \text{ porte de } \varphi} (Q_{\varphi'} \Rightarrow \psi_{\varphi'})$ sont équi-satisfiables.*

La preuve de la proposition 6.4 montre en fait que ψ est satisfaite par une extension de φ.

Démonstration. Supposons φ satisfaite par une interprétation I. On étend cette interprétation en associant, pour chaque porte φ', $[\![\varphi']\!]^I$ à la proposition $Q_{\varphi'}$: $I' \stackrel{\text{def}}{=} I[[\![\varphi']\!]^I / Q_{\varphi'}]_{\varphi'}$. Montrons que $I' \vDash \psi$ et donc que ψ est satisfiable.

Tout d'abord, comme $I \vDash \varphi$, $I' \vDash Q_\varphi$. Puis on montre par induction sur les portes φ' que $I' \vDash Q_{\varphi'} \Rightarrow \psi_{\varphi'}$. Pour le cas de base où $\varphi' = P$ (resp. $\varphi' = \neg P$), $I' \vDash Q_P \Rightarrow P$ (resp. $I' \vDash Q_P \Rightarrow \neg P$) puisque $I' \vDash Q_P$ si et seulement si $I \vDash P$ (resp. $I \vDash \neg P$). Pour l'étape d'induction,

- cas $\varphi' = \varphi_1 \wedge \varphi_2$: si $I' \vDash Q_{\varphi'}$ alors $I \vDash \varphi_1 \wedge \varphi_2$ et donc $I \vDash \varphi_1$ et $I \vDash \varphi_2$. Donc $I' \vDash Q_{\varphi_1}$ et $I' \vDash Q_{\varphi_2}$: on a bien $I' \vDash Q_{\varphi'} \Rightarrow (Q_{\varphi_1} \wedge Q_{\varphi_2})$.
- cas $\varphi' = \varphi_1 \vee \varphi_2$: on fait une analyse similaire.

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

Pour le cas de base $\varphi' = P$ (resp. $\varphi' = \neg P$), on a par hypothèse $I \vDash Q_P \Rightarrow P$ (resp. $I \vDash Q_P \Rightarrow \neg P$) et donc $I \vDash Q_P$ implique $I \vDash P$ (resp. $I \vDash \neg P$). Pour l'étape d'induction,

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

Corollaire 6.5. *Pour toute formule propositionnelle, on peut construire en temps déterministe linéaire une formule équi-satisfiable sous forme 3-clausale.*

Exemple 6.6. Reprenons la formule de l'exemple 6.3 pour $n = 3$: une formule en forme 3-clausale équi-satisfiable avec $\varphi = \bigvee_{1 \leq i \leq 3} P_i \wedge Q_i$ (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 $(R_1 \vee R_2 \vee R_3) \wedge \bigwedge_{1 \leq i \leq 3} (\neg R_i \vee P_i) \wedge (\neg R_i \vee Q_i)$.

6.3. **Forme normale disjonctive.** Toujours en utilisant la distributivité, on obtient une mise sous *forme normal disjonctive* $\mathrm{dnf}(\varphi)$ pour toute φ en forme normale négative :

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

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

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) : une conjonction $\bigwedge_{1 \leq j \leq n_i} \ell_{i,j}$ de littéraux est en effet satisfiable si et seulement si elle ne contient pas à la fois une proposition P et sa négation $\neg P$.

Comme la mise sous forme normale conjonctive, cette transformation peut avoir un coût exponentiel. Cependant, et contrairement à ce que nous venons de voir pour la forme normale conjonctive en section 6.2, si P $\neq$ NP on ne peut pas espérer avoir un algorithme en temps polynomial pour calculer une formule sous forme normale disjonctive équi-satisfiable avec une formule donnée en entrée.

7. Complétude fonctionnelle

Soit φ une formule propositionnelle. Par la propriété 2.1, la satisfaction de φ ne dépend que de l'interprétation des propositions de $\mathcal{P}_0(\varphi)$. On peut ainsi voir φ comme définissant une fonction des interprétations partielles $I \in \mathbb{B}^{\mathcal{P}_0(\varphi)}$ dans $\mathbb{B}$: $[\![\varphi]\!](I) \stackrel{\text{def}}{=} [\![\varphi]\!]^I$. C'est donc une *fonction booléenne* $\mathbb{B}^n \to \mathbb{B}$ à $n = |\mathcal{P}_0(\varphi)|$ variables.

On note $\mathcal{F}(\mathbb{B}) \stackrel{\text{def}}{=} \bigcup_{n>0} (\mathbb{B}^n \to \mathbb{B})$ l'ensemble des fonctions booléennes. Un *clone* booléen est un sous-ensemble $X \subseteq \mathcal{F}(\mathbb{B})$ tel que

(1) X contient les projections $\pi^n_k \colon \mathbb{B}^n \to \mathbb{B}$ définies par $\pi^n_k(x_1, \ldots, x_n) \stackrel{\text{def}}{=} x_k$ pour tout $n \geq k > 0$.

(2) X est fermé par composition : si $f \colon \mathbb{B}^m \to \mathbb{B}$ ainsi que $g_1, \ldots, g_m \colon \mathbb{B}^n \to \mathbb{B}$ sont dans X, alors X contient aussi $h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$.

Si B est un ensemble de fonctions de $\mathcal{F}(\mathbb{B})$, on note $[B]$ le plus petit clone qui contient B et on appelle B la *base* de $[B]$.

Pour une base $B \subseteq \mathcal{F}(\mathbb{B})$, on peut définir un ensemble de formules propositionnelles sur B en traitant chaque fonction de B à n arguments comme un symbole d'arité n : B donne lieu à une syntaxe abstraite

$$\varphi ::= f(\varphi, \ldots, \varphi) \mid P \qquad (B\text{-formules})$$

où $f \in B$ et $P \in \mathcal{P}_0$. Les fonctions sémantiques $[\![\varphi]\!]$ des B-formules engendrent alors exactement le clone $[B]$.

💬 *On peut s'intéresser à B-*SAT *le problème de satisfiabilité des B-formules.* Lewis *[1979] montre que B-*SAT *est* NP*-complet si $\not\Rightarrow \in [B]$ et est dans* P *sinon, où $x \not\Rightarrow y \stackrel{\text{def}}{=} \neg(x \Rightarrow y)$. C'est un résultat de* dichotomie*, qui montre que le théorème de* Ladner *ne s'applique pas aux problèmes expressibles comme des instances de B-*SAT*.*

7.1. **Exemples de bases complètes.** Un ensemble $B \subseteq \mathcal{F}(\mathbb{B})$ est *fonctionnellement complet* si $[B] = \mathcal{F}(\mathbb{B})$.

Propriété 7.1. *La base $\{\neg, \vee\}$ est fonctionnellement complète.*

Démonstration. Soit f une fonction booléenne $\mathbb{B}^n \to \mathbb{B}$ pour un certain $n > 0$. Si $n = 1$, alors il y a quatre fonctions booléennes de $\mathbb{B}$ dans $\mathbb{B}$, qui sont toutes dans $[\{\neg, \vee\}]$:
- la fonction identité $\pi_1^1(x) = x$,
- la fonction négation $\neg$,
- la fonction constante $\top(x) = x \vee \neg x$ et
- la fonction constante $\bot(x) = \neg(x \vee \neg x)$.

Si $n > 1$, alors $f = \bigvee_{J \in \mathbb{B}^n : f(J) = \top} f_J$ où $f_J(x_1, \ldots, x_n) = \top$ si et seulement si $(x_1, \ldots, x_n) = J$; ces dernières fonctions f_J sont dans $[\{\neg, \vee\}]$ puisque $f_J = \neg\big(\bigvee_{1 \leq i \leq n} \neg(\ell_{J,i})\big)$ où $\ell_{J,i} = x_i$ si $i \in J$ et $\ell_{J,i} = \neg x_i$ sinon. □

💬 *La formule pour f est appelée sa* forme normale disjonctive complète. *C'est manifestement une forme canonique, mais assez peu utile puisque systématiquement de taille exponentielle en n.*

Quand B est fonctionnellement complète, les B-formules fournissent une forme normale pour les formules propositionnelles : puisque $[B] = \mathcal{F}(\mathbb{B})$, $[B]$ contient en particulier deux B-formules pour $\neg$ et $\vee$. Voici deux exemples.

Soient les fonctions $\Rightarrow$ à deux arguments et $\bot$ à un argument définies par $\top \Rightarrow \bot = \bot$, $\bot \Rightarrow \top = \top \Rightarrow \top = \bot \Rightarrow \bot = \top$ et $\bot(\top) = \bot(\bot) = \bot$.

Propriété 7.2. *La base $\{\Rightarrow, \bot\}$ est fonctionnellement complète.*

Démonstration. Par la propriété 7.1, il suffit de montrer que les fonctions $\vee$ et $\neg$ appartiennent au clone $[\{\Rightarrow, \bot\}]$. On a en effet $\neg x = (x \Rightarrow \bot(x))$ pour tout $x \in \mathbb{B}$, d'où $x \vee y = \neg(x) \Rightarrow y = (x \Rightarrow \bot(x)) \Rightarrow y$ pour tous $x, y \in \mathbb{B}$. □

Soit la fonction « nor » $\downarrow$ à deux arguments définie par $\bot \downarrow \bot = \top$ et $\bot \downarrow \top = \top \downarrow \bot = \top \downarrow \top = \bot$.

Propriété 7.3. *La base $\{\downarrow\}$ est fonctionnellement complète.*

Démonstration. Par la propriété 7.1, il suffit de montrer que les fonctions $\vee$ et $\neg$ appartiennent au clone $[\{\downarrow\}]$. On a en effet $\neg(x) = x \downarrow x$ et $x \vee y = \neg(x \downarrow y) = (x \downarrow y) \downarrow (x \downarrow y)$. □

Soit enfin la fonction « xor » $\oplus$ à deux arguments définie par $\bot \oplus \bot = \top \oplus \top = \bot$ et $\bot \oplus \top = \top \oplus \bot = \top$.

Propriété 7.4. *La base $\{\oplus, \wedge, \top\}$ est fonctionnellement complète.*

Démonstration. Par la propriété 7.1, il suffit de montrer que les fonctions $\vee$ et $\neg$ appartiennent au clone $[\{\oplus, \wedge, \top\}]$. On a en effet $\neg(x) = x \oplus \top$ et $x \vee y = (x \wedge y) \oplus x \oplus y$. □

🔖 *À noter que $\oplus$ n'est jamais que l'addition des entiers modulo 2. Les polynômes multivariés sur $\mathbb{Z}/2\mathbb{Z}$ donnent ainsi une autre forme canonique pour les fonctions booléennes, appelée* forme multilinéaire, polynôme de Zhegalkin *ou encore* expansion de Reed-Muller *; voir [Goubault-Larrecq et Mackie, 1997, sec. 2.5.2] et [Knuth, 2008, pp.5–6] et songer aux leçons **120** « Anneaux $\mathbb{Z}/n\mathbb{Z}$. Applications » et **123** « Corps finis. Applications. ».*

7.2. **Exemple de base incomplète.** L'ensemble des clones booléens équipé de l'inclusion forme un treillis complet : en effet l'intersection d'une famille de clones est elle-même un clone. Ce treillis est appelé *treillis de Post*, et comprend cinq clones maximaux différents de $\mathcal{F}(\mathbb{B})$. En voici un.

Propriété 7.5. *La base $\{\wedge, \vee, \top, \bot\}$ n'est pas fonctionnellement complète ; $M \stackrel{\text{def}}{=} [\{\wedge, \vee, \top, \bot\}]$ est l'ensemble des fonctions booléennes f monotones : si $(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n)$ pour l'ordre produit sur $\mathbb{B}^n$, alors $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$.*

Démonstration. Soit f une fonction de M. On montre par induction sur une $\{\wedge, \vee, \top, \bot\}$-formule φ telle que $[\![\varphi]\!] = f$ que f est monotone. En effet, pour le cas de base, $[\![P]\!]$ est bien monotone, puis pour l'étape d'induction utilisant $\{\wedge, \vee, \top, \bot\}$ avec des sous-formules monotones, la sémantique reste monotone. Or il existe des fonctions booléennes non monotones (comme par exemple $\neg$). Donc $M \subsetneq \mathcal{F}(\mathbb{B})$.

Soit f une fonction monotone $\mathbb{B}^n \to \mathbb{B}$ pour $n > 0$. Si $n = 1$, il y a trois fonctions monotones : π_1^1, $\bot$ et $\top$. Si $n > 1$, comme $\mathbb{B}^n$ est un treillis fini pour l'ordre produit, il y a un nombre fini de tuples $J \in \mathbb{B}^n$ minimaux tels que $f(J) = \top$. Alors $f = \bigvee_{J \in \min\{J' \in \mathbb{B}^n \mid f(J') = \top\}} f_J$, où $f_J \stackrel{\text{def}}{=} \bigwedge_{1 \leq i \leq n} \ell_{J,i}$ avec $\ell_{J,i} = x_i$ si $i \in J$ et $\ell_{J,i} = \top(x_i)$ sinon ; on a en effet $f_J(x_1, \ldots, x_n) = \top$ si et seulement si $J \leq (x_1, \ldots, x_n)$. □

8. Diagrammes binaires de décision

[Goubault-Larrecq et Mackie, 1997, sec. 1.4.4], [Lassaigne et de Rougemont, 2004, sec. 1.3.5]

Pour finir cette partie 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.

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

$$\beta ::= P?\beta : \beta \mid \top \mid \bot \qquad \text{(formule de Shannon)}$$

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

Comme pour les formules propositionnelles, une formule de Shannon peut être représentée (voir figure 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 **901** « Structures de données. Exemples et applications. », voir [Knuth, 2011, sec. 7.1.4].*

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

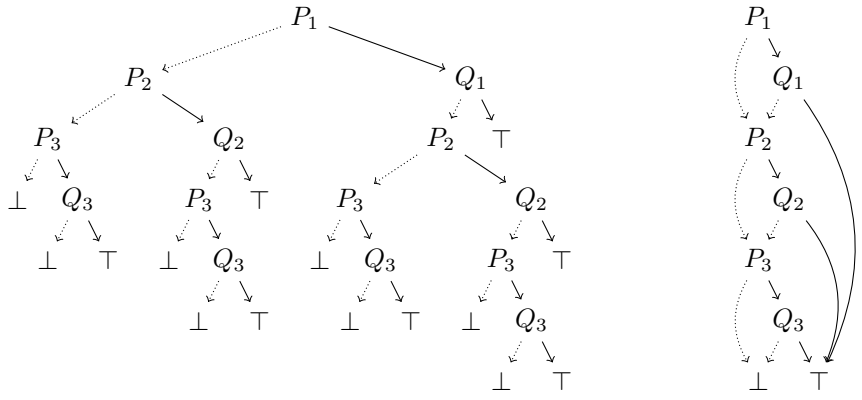

Figure 3. Un BDT et un BDD pour $(P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee (P_3 \wedge Q_3)$.

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 $P?\varphi[\top/P] : \varphi[\bot/P]$ par le lemme 3.1 de substitution propositionnelle ; il suffit d'itérer ce processus sur les propositions de $\mathcal{P}_0(\varphi)$. Le coût de cette transformation peut être exponentiel.

8.1. 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 $P?\beta_\top : \beta_\bot$ et pour toute proposition $Q \in \mathcal{P}_0(\beta_\top) \cup \mathcal{P}_0(\beta_\bot)$, $P < Q$. 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 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.

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* $\mathsf{P} \neq \mathsf{NP}$ *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 et al., 1995, thm. 3].*

8.2. **Formules de Shannon complètes.** Un cas particulier des formule de Shannon ordonnées est celui des formules *complètes* sur un ensemble de propositions $P_1 < \cdots < P_n$: toutes ces propositions apparaissent dans l'ordre dans toutes les branches. Un OBDD complet est aussi appelé un *QDD*. La figure 4 montre le QDD correspondant à la formule de la figure 3.

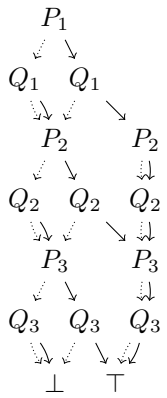

Figure 4. Le QDD pour $(P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee (P_3 \wedge Q_3)$.

Un QDD β sur $P_1 < \cdots < P_n$ est essentiellement isomorphe à l'automate déterministe complet minimal pour $L(\beta) \stackrel{\text{def}}{=} \{I \in \mathbb{B}^n \mid I \vDash \beta\}$ – aux transitions sortantes des sommets $\bot$ et $\top$ près [Wegener, 2000, sec. 3.2]. Cela montre que la représentation par QDD est elle aussi canonique.

Proposition 8.1 (borne supérieure). *Soit β un QDD sur n propositions. Alors $|\beta| \leq \frac{2^n}{n}$.*

[Wegener, 2000, thm. 2.3.1].

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 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 $2^{n-k}-1$.

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 :

$$\begin{aligned}
|\beta_k| &= 2^{n-\lfloor \lg(n-\lg n)\rfloor} + 2^{\lfloor \lg(n-\lg n)\rfloor} \cdot 2^{2^{\lfloor \lg(n-\lg n)\rfloor}} \\
&\leq 2^{n-\lfloor \lg(n-\lg n)\rfloor} + 2^{\lfloor \lg(n-\lg n)\rfloor} \cdot 2^{n-\lg n} && \text{(car } 2^x \text{ est croissante)} \\
&= 2^n \cdot \left(2^{-\lfloor \lg(n-\lg n)\rfloor} + 2^{\lfloor \lg(n-\lg n)\rfloor - \lg n}\right) \\
&\leq 2^n \cdot \left(2^{-\lfloor \lg(n-\lg n)\rfloor} \cdot 2^{\lfloor \lg(n-\lg n)\rfloor - \lg n}\right) && \text{(car } 2^x \text{ est convexe)} \\
&= 2^n \cdot 2^{-\lg n} \,. && \square
\end{aligned}$$

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 $\lim_{n\to\infty} \varepsilon_n = 0$ [Breitbart et al., 1995, thm. 1], montrant que la borne supérieure de la proposition 8.1 est optimale.

8.3. **Formules de Shannon réduites.** Une formule de Shannon est *réduite* si pour toute sous-formule $P?\beta_\top : \beta_\bot$, les formules $\beta_\top$ et $\beta_\bot$ 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 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 $\bot$.

Le principal résultat dans le cadre de la leçon **916** est l'unicité des ROBDD.

[Goubault-Larrecq et Mackie, 1997, thm. 2.48], [Lassaigne et de Rougemont, 2004, cor. 1.4], [Wegener, 2000, thm. 3.1.4 et 3.3.2]

Théorème 8.2 (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.*

Démonstration. Si β et β' sont égaux, alors ils sont clairement équivalents. Pour la réciproque, on montre par récurrence sur $|\mathcal{P}_0(\beta)| + |\mathcal{P}_0(\beta')|$ 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 $\bot$ ou $\top$.

Pour l'étape de récurrence, soit P la proposition minimale de $\mathcal{P}_0(\beta) \cup \mathcal{P}_0(\beta')$. Sans perte de généralité, on va supposer que $P \in \mathcal{P}_0(\beta)$. Comme $P = \min_< \mathcal{P}_0(\beta)$, $\beta = P?\beta_\top : \beta_\bot$.

- **Si $P \notin \mathbf{fv}(\beta')$:** alors β', $\beta_\top$ et $\beta_\bot$ sont logiquement équivalents, ce qui contredit le fait que β était réduit. En effet, si $I \vDash \beta'$, alors par la propriété 2.1, $I[\top/P] \vDash \beta'$. Par équivalence de β et β', $I[\top/P] \vDash \beta$, et donc $I[\top/P] \vDash \beta_\top$ et comme $P \notin \mathcal{P}_0(\beta_\top)$ par la propriété 2.1 $I \vDash \beta_\top$. Inversement, si $I \vDash \beta_\top$, alors $I[\top/P] \vDash \beta_\top$ par la propriété 2.1, donc $I[\top/P] \vDash \beta$; comme $\beta \equiv \beta'$, $I[\top/P] \vDash \beta'$ et par la propriété 2.1 $I \vDash \beta$. Le même raisonnement vaut pour $\beta_\bot$.
- **Si $P \in \mathbf{fv}(\beta')$:** alors $\beta' = P?\beta'_\top : \beta'_\bot$ et on observe que $\beta_\top$ et $\beta'_\top$ sont équivalents et $\beta_\bot$ et $\beta'_\bot$ aussi ; par hypothèse de récurrence on a alors $\beta_\top = \beta'_\top$ et $\beta_\bot = \beta'_\bot$ et donc $\beta = \beta'$.

  Détaillons l'observation pour le cas de $\beta'_\top$. Si $I \vDash \beta_\top$, comme $P \notin \mathcal{P}_0(\beta_\top)$, par la propriété 2.1, $I[\top/P] \vDash \beta_\top$ et donc $I[\top/P] \vDash \beta$. Par suite, $I[\top/P] \vDash \beta'$, donc $I[\top/P] \vDash \beta'_\top$ et $P \notin \mathcal{P}_0(\beta'_\top)$, $I \vDash \beta'_\top$. La direction opposée, tout comme le cas de $\beta_\bot \equiv \beta'_\bot$, est similaire. □

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 4 construit le ROBDD de la figure 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].

Partie 3. Satisfiabilité

Le problème de *satisfiabilité* est le problème de décision suivant.

*[Knuth, 2008, sec. 7.1.1] et [Knuth, 2015, sec. 7.2.2.2] pour une présentation des aspects algorithmiques du problème de satisfiabilité. Cette partie peut aussi vous inspirer pour les leçons **915** « Classes de complexité. Exemples. » et **928** « Problèmes NP-complets : exemples et réductions ».*

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 $I \in \mathbb{B}^{\mathcal{P}_0(\varphi)}$ telle que $I \vDash \varphi$. Une telle interprétation est de taille polynomiale, et vérifier $I \vDash \varphi$ se fait en temps polynomial. La difficulté est de montrer que SAT est NP-difficile.

Théorème 8.3 (Cook-Levin)**.** SAT *est* NP*-complet.*

[Perifel, 2014, thm. 3-V], [Arora et Barak, 2009, thm. 2.10], [Papadimitriou, 1993, thm. 8.2], [Carton, 2008, thm. 4.19], [Lassaigne et de Rougemont, 2004, thm. 11.1]

La preuve de ce théorème est normalement faite dans la leçon **928** « Problèmes NP-complets : exemples et réductions ». À noter que la borne inférieure vaut déjà pour une représentation arborescente, tandis que la borne supérieure vaut aussi pour une représentation sous forme de circuit (*c.f.* figure 1).

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 kSAT.

Problème (kSAT)**.**
instance : un ensemble fini F de k-clauses propositionnelles
question : F est-il satisfiable ?

Théorème 8.4. kSAT *est* NP*-complet pour tout* $k \geq 3$.

Démonstration. C'est une conséquence immédiate du théorème de Cook-Levin combiné au corollaire 6.5. □

Dans cette partie, nous allons tout d'abord considérer dans les sections 9 et 10 comment des systèmes de preuve peuvent être employés pour résoudre SAT. Cela fournit des algorithmes à base de *recherche de preuve*, plus « structurés » qu'une énumération brutale des interprétations possibles. En chemin, cela permettra aussi de démontrer deux théorèmes de complétude (théorème 9.6 pour le calcul des séquents propositionnels et théorème 10.12 pour la résolution propositionnelle) et de voir une troisième preuve du théorème de compacité (section 10.1.2).

Dans un deuxième temps, nous allons voir deux variantes plus simples de SAT : le problème P-complet HornSAT en section 12 et le problème NL-complet 2SAT en section 13.

9. Calcul des séquents propositionnel

[Goubault-Larrecq et Mackie, 1997, fig. 2.2]

Le premier système de preuve de ces notes de révision 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* sans coupure ni contraction, 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 $m \colon S \to \mathbb{N}$ de domaine $\mathrm{dom}(m) \stackrel{\text{def}}{=} \{e \in S \mid m(e) > 0\}$ fini ; on peut aussi voir m comme une séquence finie de S^* modulo permutation.

Dans le calcul de la figure 5, un *séquent* est un multi-ensemble fini de formules en forme normale négative 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 φ, et $\Gamma(\psi) + \Delta(\psi)$ occurrences de $\psi \neq \varphi$. On note le séquent vide $\vdash \bot$, où $\bot(\varphi) \stackrel{\text{def}}{=} 0$ pour toute formule φ.

Une règle du calcul des séquents permet de déduire un séquent *conclusion* d'un nombre fini de séquents *prémisses*. Chaque règle comprend une *formule principale* dans sa conclusion, indiquée

💬 *Ce calcul ne contient pas les règles de* coupure *ni de* contraction*, mais celles-ci sont admissibles ; voir les notes pour la leçon* ***918****.*

$$\frac{}{\vdash \Gamma, \varphi, \overline{\varphi}}\ (\mathsf{ax}) \qquad \frac{\vdash \Gamma, \varphi \quad \vdash \Gamma, \psi}{\vdash \Gamma, \varphi \wedge \psi}\ (\wedge) \qquad \frac{\vdash \Gamma, \varphi, \psi}{\vdash \Gamma, \varphi \vee \psi}\ (\vee)$$

FIGURE 5. Calcul des séquents propositionnel monolatère.

en orange dans les règles de la figure 5. Un séquent $\vdash \Gamma$ est *prouvable*, noté $\vdash_{\mathbf{LK}_0} \Gamma$, s'il en existe une dérivation dans le système de la figure 5.

Exemple 9.1. La loi de PIERCE de l'exemple 5.1 est prouvable. La dérivation correspondante (avec la formule principale indiquée en orange à chaque étape) est :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg P, Q, P}\,(\mathsf{ax})}{\vdash \neg P \vee Q, P}\,(\vee) \qquad \dfrac{}{\vdash \neg P, P}\,(\mathsf{ax})}{\vdash (\neg P \vee Q) \wedge \neg P, P}\,(\wedge)}{\vdash ((\neg P \vee Q) \wedge \neg P) \vee P}\,(\vee)$$

9.1. **Recherche de preuve.** La recherche de preuve en calcul des séquents propositionnel vise à répondre au problème de décision suivant.

Problème (PROUVABILITÉ)**.**
instance : un séquent propositionnel $\vdash \Gamma$
question : $\vdash \Gamma$ est-il prouvable dans le calcul des séquents propositionnel ?

La recherche de preuve tente de développer un arbre de dérivation avec $\vdash \Gamma$ pour conclusion, de la racine vers les feuilles. Une *branche* de recherche de preuve est une séquence potentiellement infinie de séquents $\vdash \Gamma = \vdash \Gamma_0, \vdash \Gamma_1, \ldots$ calculée par la recherche de preuve : pour tout $i > 0$, $\vdash \Gamma_i$ est une prémisse d'une règle dont $\vdash \Gamma_{i-1}$ est la conclusion. Une *branche d'échec* est une branche de recherche de preuve finie $\vdash \Gamma_0, \vdash \Gamma_1, \ldots, \vdash \Gamma_n$ où aucune règle ne s'applique à $\vdash \Gamma_n$, c'est-à-dire qu'il n'est le séquent conclusion d'aucune règle.

On peut montrer que les branches de recherche de preuve dans le calcul des séquents propositionnel de la figure 5 sont toutes finies. On définit pour cela la *profondeur* d'un multi-ensemble Γ comme la somme des profondeurs de ses occurrences de formules $p(\Gamma) \stackrel{\text{def}}{=} \sum_{\varphi \in \mathrm{dom}(\Gamma)} p(\varphi) \cdot \Gamma(\varphi)$.

Propriété 9.2 (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 $p(\Gamma)$.*

Démonstration. On peut vérifier que pour chacune des règles (ax), (∨) et (∧), la profondeur de chaque séquent prémisse est strictement inférieure à celle du séquent conclusion. □

La recherche de preuve n'est pas déterministe : il peut y avoir plusieurs règles applicables pour un même séquent $\vdash \Gamma$. Par exemple, pour $\vdash \neg P \vee Q, R \vee P$, on peut commencer par sélectionner $\neg P \vee Q$ comme formule principale et appliquer (∨) :

$$\frac{\vdash \neg P, Q, R \vee P}{\vdash \neg P \vee Q, R \vee P}\ (\vee)$$

Mais on aurait aussi pu sélectionner $R \vee P$ comme formule principale et appliquer (∨) :

$$\frac{\vdash \neg P \vee Q, R, P}{\vdash \neg P \vee Q, R \vee P}\ (\vee)$$

En général – mais comme nous allons le voir, pas dans le calcul des séquents de la figure 5 – il est nécessaire pour dans un algorithme déterministe de recherche de preuve de mémoriser de tels *points de choix* pour pouvoir y revenir par *backtracking* en cas d'échec.

Une règle de déduction est *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 inversible, mais ce qui est plus intéressant, c'est que toutes les autres règles de la figure 5 le sont aussi. 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 9.3 (inversibilité)**.** *Les règles du calcul des séquents propositionnel sont inversibles.*

Démonstration. Comme déjà mentionné, la règle (ax) est 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 $\vdash_{\mathbf{LK}_0} \Gamma, \varphi, \psi$.

- Si π se termine par ($\vee$) où $\varphi \vee \psi$ est principale, alors évidemment il existe une sous-dérivation de π pour sa prémisse $\vdash \Gamma, \varphi, \psi$.
- Si π se termine par (ax) où $\varphi \vee \psi$ est principale, alors $\Gamma = \Gamma', \overline{\varphi} \wedge \overline{\psi}$ et on a la dérivation

$$\dfrac{\dfrac{}{\vdash \Gamma', \varphi, \psi, \overline{\varphi}}\,(\text{ax}) \qquad \dfrac{}{\vdash \Gamma', \varphi, \psi, \overline{\psi}}\,(\text{ax})}{\vdash \Gamma', \varphi, \psi, \overline{\varphi} \wedge \overline{\psi}}\,(\wedge)$$

- 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

$$\dfrac{\begin{array}{ccc}\pi_1 & & \pi_k \\ \vdots & & \vdots \\ \vdash \Gamma_1, \varphi \vee \psi & \cdots & \vdash \Gamma_k, \varphi \vee \psi\end{array}}{\vdash \Gamma, \varphi \vee \psi}\,(\text{R})$$

  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

$$\dfrac{\begin{array}{ccc}\pi_1' & & \pi_k' \\ \vdots & & \vdots \\ \vdash \Gamma_1, \varphi, \psi & \cdots & \vdash \Gamma_k, \varphi, \psi\end{array}}{\vdash \Gamma, \varphi, \psi}\,(\text{R})$$

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 $\vdash_{\mathbf{LK}_0} \Gamma, \varphi$ et $\vdash_{\mathbf{LK}_0} \Gamma, \psi$.

- 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 $\vdash \Gamma, \psi$.
- Si π se termine par (ax) où $\varphi \wedge \psi$ est principale, alors $\Gamma = \Gamma', \overline{\varphi} \vee \overline{\psi}$ et on a les dérivations

$$\dfrac{\dfrac{}{\vdash \Gamma', \overline{\varphi}, \overline{\psi}, \varphi}\,(\text{ax})}{\vdash \Gamma', \overline{\varphi} \vee \overline{\psi}, \varphi}\,(\vee) \qquad \text{et} \qquad \dfrac{\dfrac{}{\vdash \Gamma', \overline{\varphi}, \overline{\psi}, \psi}\,(\text{ax})}{\vdash \Gamma', \overline{\varphi} \vee \overline{\psi}, \psi}\,(\vee)$$

- 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

$$\dfrac{\begin{array}{ccc}\pi_1 & & \pi_k \\ \vdots & & \vdots \\ \vdash \Gamma_1, \varphi \wedge \psi & \cdots & \vdash \Gamma_k, \varphi \wedge \psi\end{array}}{\vdash \Gamma, \varphi \wedge \psi}\,(\text{R})$$

  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

$$\dfrac{\begin{array}{ccc}\pi'_1 & & \pi'_k \\ \vdots & & \vdots \\ \vdash \Gamma_1, \varphi & \cdots & \vdash \Gamma_k, \varphi\end{array}}{\vdash \Gamma, \varphi}\ (\mathrm{R}) \qquad \text{et} \qquad \dfrac{\begin{array}{ccc}\pi''_1 & & \pi''_k \\ \vdots & & \vdots \\ \vdash \Gamma_1, \psi & \cdots & \vdash \Gamma_k, \psi\end{array}}{\vdash \Gamma, \psi}\ (\mathrm{R})$$

□

Corollaire 9.4. 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 9.2, 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 approximation un algorithme en AP = PSPACE.

Cependant, par le lemme 9.3, 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. □

9.2. **Correction et complétude.** Un séquent $\vdash \Gamma$ est *satisfait* par une interprétation I, ce qu'on écrit $I \vDash \Gamma$, s'il existe une formule témoin $\vartheta \in \mathrm{dom}(\Gamma)$ telle que $I \vDash \vartheta$. On dit qu'un séquent $\vdash \Gamma$ est *valide* et on écrit $\vDash \Gamma$ si pour toute interprétation I, $I \vDash \Gamma$. La correction et la complétude du calcul des séquents montrent qu'un séquent est prouvable si et seulement s'il est valide.

Théorème 9.5 (correction). *Si* $\vdash_{\mathbf{LK}_0} \Gamma$, *alors* $\vDash \Gamma$.

Démonstration. On procède par induction structurelle 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 $\Gamma = \Gamma', \varphi, \overline{\varphi}$ pour une formule φ. Pour toute interprétation I, soit $I \vDash \varphi$, soit $I \vDash \overline{\varphi}$; dans tous les cas $I \vDash \Gamma$.
- **Pour ($\vee$) :** alors $\Gamma = \Gamma', \varphi \vee \psi$ où $\vdash_{\mathbf{LK}_0} \Gamma', \varphi, \psi$. Soit I une interprétation quelconque ; alors par hypothèse d'induction $I \vDash \Gamma', \varphi, \psi$. Il existe donc une formule témoin $\vartheta \in \mathrm{dom}(\Gamma)' \cup \{\varphi, \psi\}$ telle que $I \vDash \vartheta$. Si $\vartheta \in \{\varphi, \psi\}$, alors $I \vDash \varphi \vee \psi$, sinon $\vartheta \in \mathrm{dom}(\Gamma)'$ et alors $I \vDash \Gamma'$; dans tous les cas $I \vDash \Gamma$.
- **Pour ($\wedge$) :** alors $\Gamma = \Gamma', \varphi \wedge \psi$ où $\vdash_{\mathbf{LK}_0} \Gamma', \varphi$ et $\vdash_{\mathbf{LK}_0} \Gamma', \psi$. Soit I une interprétation quelconque ; alors par hypothèse d'induction $I \vDash \Gamma', \varphi$ et $I \vDash \Gamma', \psi$. Il existe donc une formule témoin $\vartheta \in \mathrm{dom}(\Gamma)' \cup \{\varphi\}$ telle que $I \vDash \vartheta$ et une formule témoin $\vartheta' \in \mathrm{dom}(\Gamma)' \cup \{\psi\}$ telle que $I \vDash \vartheta'$. Si $\vartheta \in \mathrm{dom}(\Gamma)'$ ou $\vartheta' \in \mathrm{dom}(\Gamma)'$, alors $I \vDash \Gamma'$. Sinon, $\vartheta = \varphi$ et $\vartheta' = \psi$ et donc $I \vDash \varphi \wedge \psi$. Dans tous les cas $I \vDash \Gamma$. □

Théorème 9.6 (complétude). *Si* $\vDash \Gamma$, *alors* $\vdash_{\mathbf{LK}_0} \Gamma$.

Démonstration. On procède par contraposition : on suppose $\vdash \Gamma$ non prouvable, et on montre que $\vdash \Gamma$ n'est pas valide, en exhibant une interprétation I telle que $I \nvDash \Gamma$.

On commence par observer que si $\vdash \Gamma$ n'est pas prouvable, alors il existe une branche d'échec. En effet, par la propriété 9.2, les branches de recherche de preuve sont finies, donc elles terminent toutes soit par un séquent pour lequel aucune règle n'est applicable, 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 aucune règle ne s'applique à $\vdash \Gamma_n$. Comme ni $(\vee)$ ni $(\wedge)$ ne s'applique, $\mathrm{dom}(\Gamma)_n$ ne contient que des littéraux : $\mathrm{dom}(\Gamma) = \{\ell_1, \dots, \ell_k\}$. Comme (ax) ne s'applique pas non plus, $\mathrm{dom}(\Gamma)_n$ ne contient pas un littéral et sa négation ; donc il existe des interprétations I telles que $I \not\models \ell_j$ pour tout $1 \leq j \leq k$, c'est-à-dire telles que $I \not\models \ell_1, \dots, \ell_k$.

On montre par récurrence décroissante sur $n \geq i \geq 0$ que si I est une interprétation telle que $I \not\models \ell_1, \dots, \ell_k$, alors $I \not\models \Gamma_i$. Pour le cas de base au rang $i = n$, clairement $I \not\models \Gamma_n$ puisque $\mathrm{dom}(\Gamma)_n = \{\ell_1, \dots, \ell_k\}$. Pour l'étape de récurrence au rang $i - 1$, on fait une distinction de cas selon la règle qui a permis de passer de Γ_{i-1} à Γ_i dans la branche d'échec.

Cas de (ax) : ce cas est impossible puisque c'est une branche d'échec.

Cas de $(\vee)$: alors $\Gamma_i = \Gamma', \varphi, \psi$ et $\Gamma_{i-1} = \Gamma', \varphi \vee \psi$. Soit I telle que $I \not\models \ell_1, \dots, \ell_k$; par hypothèse de récurrence, $I \not\models \Gamma', \varphi, \psi$. 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 donc, puisque $I \not\models \Gamma'$, que $I \not\models \Gamma_{i-1}$.

Cas de $(\wedge)$: alors on peut supposer que $\Gamma_i = \Gamma', \varphi$ et $\Gamma_{i-1} = \Gamma', \varphi \wedge \psi$ (puisque le cas où $\Gamma_i = \Gamma', \psi$ est similaire). Soit I telle que $I \not\models \ell_1, \dots, \ell_k$; par hypothèse de récurrence, $I \not\models \Gamma', \varphi$. 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 $I \not\models \Gamma_{i-1}$.

Pour conclure, on a montré qu'il existait des interprétations I telles que $I \not\models \ell_1, \dots, \ell_k$, et que pour une telle interprétation, $I \not\models \Gamma$. Donc $\not\models \Gamma$. □

Corollaire 9.7. SAT *est dans* NP.

Démonstration. Soit φ une formule propositionnelle. Elle est satisfiable si et seulement si $\neg\varphi$ n'est pas valide. On calcule alors en temps déterministe polynomial sa forme normale négative $\overline{\varphi}$ qui n'est pas valide si et seulement si $\neg\varphi$ n'est pas valide. Par les théorèmes de correction et de complétude, $\overline{\varphi}$ n'est pas valide si et seulement si $\vdash \overline{\varphi}$ n'est pas prouvable. On fait une recherche de preuve pour $\vdash \overline{\varphi}$, qui résout la prouvabilité en coNP par le corollaire 9.4. L'algorithme en entier est donc en NP. □

L'intérêt des corollaires 9.7 et 9.8 ne réside pas dans le fait que SAT *soit dans* NP *– il y en a des preuves bien plus simples ! Ils montrent que la recherche de preuve en calcul des séquents a une complexité « optimale » pour résoudre* SAT*, l'optimalité s'entendant avec la granularité assez grossière de la classe de complexité* coNP.

Corollaire 9.8. PROUVABILITÉ *est* coNP*-complet.*

Démonstration. Par le corollaire 9.4, PROUVABILITÉ est dans coNP. Par les théorèmes de correction et complétude, une formule propositionnelle φ est valide si et seulement si le séquent propositionnel $\vdash \varphi$ est prouvable, ce qui fournit une réduction triviale depuis le problème VALIDITÉ, qui est coNP-complet puisque c'est le dual de SAT. □

10. Résolution propositionnelle

Le second système de preuve de ces notes de révision est le système de résolution propositionnelle. L'idée de ce système de preuve est de montrer l'insatisfiabilité de $\neg\varphi$ plutôt que la validité de φ, en dérivant « $\bot$ » depuis $\neg\varphi$.

[Goubault-Larrecq et Mackie, 1997, sec. 2.4.2]

La résolution a l'intérêt d'avoir des preuves de réfutation finies*, même pour des ensembles infinis de clauses.*

$$\frac{C, P \quad \neg P, D}{C, D}\ (\mathsf{res}) \qquad \frac{}{P, \neg P}\ (\mathsf{te}) \qquad \frac{C}{C, \ell}\ (\mathsf{aff})$$

Figure 6. Règles complètes de la résolution propositionnelle.

Le système travaille sur des formules sous forme clausale. Le système de la figure 6 représente chaque clause $\ell_1 \vee \cdots \vee \ell_n$ comme un ensemble fini de littéraux $\{\ell_1, \dots, \ell_n\}$. Les symboles C et D dénotent de tels ensembles finis, et la virgule dénote l'union ; on notera l'ensemble vide de littéraux « $\bot$ ». Le *résolvant* de la règle (res) est la proposition P indiquée en violet.

La règle (res) *applique implicitement la règle de factorisation, puisque* $C, \ell, \ell = C, \ell$.

On peut montrer des bornes exponentielles sur la taille des preuves en résolution, voir [Knuth, 2015, pp. 57–59].

Pour un ensemble S de clauses et une clause C, on note $S \vdash_{\mathbf{R}_0} C$ si C peut être dérivée à partir de clauses de S par la seule règle de résolution (res). On écrit $S \vdash_{\mathbf{R}'_0} C$ si C peut être dérivée avec les trois règles (res), (te) et (aff) depuis S.

Exemple 10.1. Comme vu dans l'exemple 6.2, la *négation* de la loi de Pierce s'écrit comme l'ensemble de clauses $S = \{P \vee P,\ \neg Q \vee P,\ \neg P\}$. Les ensembles de littéraux correspondants sont donc $\{P\}$, $\{\neg Q, P\}$ et $\{\neg P\}$ et on peut montrer $S \vdash_{\mathbf{R}_0} \bot$ par

$$\dfrac{P \qquad \neg P}{\bot}\ \text{(res)}$$

Exemple 10.2. Pour un exemple plus intéressant, considérons à nouveau l'ensemble de clauses de l'exemple 2.3 :

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

Cet ensemble dérive la clause vide :

$$\dfrac{\dfrac{\dfrac{Q, R \qquad P, Q, \neg R}{P, Q}\ \text{(res)} \qquad P, \neg Q}{P}\ \text{(res)} \qquad \dfrac{\dfrac{Q, R \qquad \neg P, \neg Q, R}{\neg P, R}\ \text{(res)} \qquad \neg P, \neg R}{\neg P}\ \text{(res)}}{\bot}\ \text{(res)}$$

Théorème 10.3 (correction)**.** *Soit S un ensemble de clauses et C une clause. Si $S \vdash_{\mathbf{R}'_0} C$ alors $S \models C$.*

Démonstration. On procède par induction sur la dérivation de $S \vdash_{\mathbf{R}'_0} C$. Pour le cas de base, $C \in S$ et donc $S \models C$. Pour l'étape d'induction, on fait une distinction de cas selon la dernière règle appliquée :

- **Pour (res) :** supposons $S \vdash_{\mathbf{R}'_0} C, P$ et $S \vdash_{\mathbf{R}'_0} \neg P, D$. Soit I une interprétation telle que $I \models S$. Par hypothèse d'induction, $I \models C, P$ donc $I \models \ell$ pour un littéral $\ell \in C \cup \{P\}$; de même $I \models \ell'$ pour un littéral $\ell' \in \{\neg P\} \cup D$. Alors $(\ell, \ell') \neq (P, \neg P)$ donc $I \models C, D$.
- **Pour (te) :** $P \vee \neg P$ est valide.
- **Pour (aff) :** supposons $S \vdash_{\mathbf{R}'_0} C$. Soit I une interprétation telle que $I \models S$. Par hypothèse d'induction, $I \models C$ et donc $I \models C, \ell$ pour tout littéral ℓ. □

10.1. **Complétude réfutationnelle.** Nous allons montrer tout d'abord la complétude réfutationnelle. Nous allons voir deux preuves de ce résultat, l'une s'appuyant sur les arbres sémantiques de la section 4.2, l'autre sur un lemme de saturation ; à noter que cette première preuve donne un résultat un peu plus fort, puisqu'elle n'utilise que la règle (res).

[Goubault-Larrecq et Mackie, 1997, p. 50]

10.1.1. *Complétude réfutationnelle par les arbres sémantiques.* Cette preuve s'appuie sur la définition des *arbres sémantiques* vue en section 4.2. On peut déjà observer que l'arbre de dérivation de l'exemple 10.2 est *isomorphe* à l'arbre sémantique fermé de la figure 2, au sens suivant : leur structure est la même, les étiquettes de leurs feuilles sont les mêmes et les étiquettes des nœuds internes de l'arbre sémantique sont les résolvants des applications de (res) dans la dérivation.

Dans un arbre sémantique fermé, on appelle *nœud d'inférence* un nœud dont les deux fils sont des nœuds d'échec.

La propriété 10.4 peut être admise dans un développement sur le théorème 10.6 de complétude réfutationnelle de la résolution.

Propriété 10.4. *Si un arbre sémantique fermé n'est pas réduit à sa seule racine, alors il contient un nœud d'inférence.*

Démonstration. Cela vaut en réalité pour tout arbre binaire fini non réduit à sa seule racine : il contient au moins un nœud dont les deux fils sont des feuilles. Cela se vérifie par récurrence sur la taille N de l'arbre. Pour le cas de base au rang $N = 3$, les deux nœuds fils de la racine sont des feuilles. Pour l'étape de récurrence au rang $N > 3$, l'un des deux fils de la racine n'est pas une feuille, et enracine un sous-arbre de taille au plus $N - 2$, donc par hypothèse d'induction ce sous-arbre contient au moins un nœud dont les deux fils sont des feuilles. □

Lemme 10.5. *Pour tout ensemble S insatisfiable de clauses et pour tout arbre sémantique fermé de S, il existe une dérivation isomorphe de $S \vdash_{\mathbf{R}_0} \bot$.*

Démonstration. Procédons par récurrence sur le nombre de nœuds de l'arbre sémantique fermé.

Pour le cas de base, si l'arbre est réduit à sa seule racine, alors celle-ci est associée à l'interprétation de domaine vide et étiquetée par une clause témoin $C \in S$, laquelle est donc falsifiée par toute interprétation. Ce ne peut être que la clause vide $\bot$, pour laquelle on a l'arbre de dérivation pour $S \vdash_{\mathbf{R}_0} \bot$ réduit à la seule clause $\bot \in S$.

Pour l'étape de récurrence, supposons l'arbre non réduit à sa seule racine. Par la propriété 10.4, il contient un nœud d'inférence, étiqueté par une proposition P et dont les deux nœuds fils sont des nœuds d'échec étiquetés respectivement par $C_1 \in S$ et $C_2 \in S$. Soit I l'interprétation partielle du nœud d'inférence : alors

$$I[\bot/P] \not\models C_1 \qquad \text{et} \qquad I[\top/P] \not\models C_2 \ . \tag{†}$$

Cependant, le nœud d'inférence n'est pas un nœud d'échec, donc $I \models C_1$ et $I \models C_2$.

Cela signifie qu'il y a un littéral de C_1 qui est satisfait par I mais pas par $I[\bot/P]$; ce littéral est donc P. De même, C_2 contient le littéral $\neg P$. Dès lors, $\{C_1, C_2\} \vdash_{\mathbf{R}_0} C$ pour la clause $C \stackrel{\text{def}}{=} C_1 \setminus \{P\}, C_2 \setminus \{\neg P\}$ par résolution sur P et donc $S \vdash_{\mathbf{R}_0} C$. On observe que $I \not\models C$: en effet, $I \not\models C_1 \setminus \{P\}$ et $I \not\models C_2 \setminus \{\neg P\}$, sans quoi on aurait $I[\bot/P] \models C_1$ ou $I[\top/P] \models C_2$ en contradiction avec (†).

On applique le même raisonnement à tous les nœuds d'inférence de l'arbre, et on considère ceux pour lesquels la même clause C a été construite. Tous ces nœuds deviennent des nœuds d'échec de $S \cup \{C\}$ étiquetés par la clause témoin C. L'arbre ainsi construit est un arbre sémantique fermé pour $S \cup \{C\}$ strictement plus petit que celui associé à S, donc par hypothèse de récurrence il existe une dérivation de $S \cup \{C\} \vdash_{\mathbf{R}_0} \bot$ isomorphe au nouvel arbre. On en conclut que $S \vdash_{\mathbf{R}_0} \bot$ par une dérivation isomorphe à l'arbre initial. □

Théorème 10.6 (complétude réfutationnelle de la résolution)**.** *Soit S un ensemble de clauses. Si S est insatisfiable alors $S \vdash_{\mathbf{R}_0} \bot$.*

Démonstration. C'est une conséquence directe du lemme 10.5 : en effet, tout ensemble insatisfiable S de clauses a un arbre sémantique fermé, et donc une réfutation $S \vdash_{\mathbf{R}_0} \bot$. □

Le lemme 10.5 montre plus que la seule complétude réfutationnelle. Comme un arbre sémantique fermé ne peut pas avoir deux fois la même proposition le long d'une même branche, cela signifie que l'on peut restreindre l'utilisation de la règle (res) de la même manière. Une dérivation de $S \vdash_{\mathbf{R}_0} \bot$ est *non redondante* si l'on n'utilise pas deux fois le même résolvant le long d'une branche – et est donc de taille au plus $2^{|\mathcal{P}_0(S)|+1}$.

Corollaire 10.7 (dérivations non redondantes)**.** *Soit S un ensemble insatisfiable de clauses. Alors il existe une dérivation non redondante de $S \vdash_{\mathbf{R}_0} \bot$.*

Dans le cas des dérivations non redondantes, on peut montrer une réciproque au lemme 10.5.

Lemme 10.8. *Soit S un ensemble insatisfiable de clauses. Pour toute dérivation non redondante de $S \vdash_{\mathbf{R}_0} \bot$, il existe un arbre sémantique fermé isomorphe pour S.*

Démonstration. On associe à chaque nœud interne de la dérivation non redondante le résolvant correspondant. Montrons que cet étiquetage est celui d'un arbre sémantique fermé pour S. Il suffit pour cela de montrer que, pour toute feuille, la clause qui étiquette la feuille est falsifiée par l'interprétation associée à la feuille. Considérons une telle feuille et soit $C = \{\ell_1, \ldots, \ell_n\}$ sa clause. Alors, puisque la dérivation termine par la clause vide à sa racine, chacun des littéraux $\ell_1, \ldots, \ell_n$ a été utilisé comme résolvant dans la branche. Donc l'interprétation partielle I associée à la feuille est telle que $I \not\models \ell_i$ pour tout $1 \leq i \leq n$, et on a bien $I \not\models C$. □

10.1.2. *Compacité par complétude réfutationnelle.* Voici une dernière preuve du théorème de compacité, qui s'appuie sur la correction et la complétude réfutationnelle de la résolution.

Démonstration du théorème de compacité. Si $S \vDash \bot$, alors l'ensemble $\mathrm{Cl}(S) \stackrel{\text{def}}{=} \bigcup_{\varphi \in S} \mathrm{Cl}(\mathrm{nnf}(\varphi))$ de ses formules mises sous forme clausale est aussi insatisfiable par la propriété 6.1 : $\mathrm{Cl}(S) \vDash \bot$. Par complétude réfutationnelle de la résolution, $\mathrm{Cl}(S) \vdash_{\mathbf{R}_0} \bot$. Mais une telle dérivation de $\bot$ à partir de clauses dans $\mathrm{Cl}(S)$ est finie ne peut utiliser qu'un ensemble fini de clauses : on a déjà $\mathrm{Cl}(F) \vdash_{\mathbf{R}_0} \bot$ pour un sous-ensemble $F \subseteq_{\text{fin}} S$. Par correction, $\mathrm{Cl}(F) \vDash \bot$, et par la propriété 6.1, $F \vDash \bot$. □

[David et al., 2003, thm. 7.4.9]

10.1.3. *Complétude réfutationnelle par saturation.* Un ensemble de clauses S est dit *saturé* si, pour toute proposition P de $\mathcal{P}_0$, soit la clause P soit la clause $\neg P$ appartient à S. On utilise dans la preuve suivante un lemme dit de *déduction* que nous verrons en section 10.2 ci-dessous.

Lemme 10.9 (saturation)**.** *Soit S un ensemble de clauses tel que $S \nvdash_{\mathbf{R}'_0} \bot$. Alors il existe un ensemble de clauses $S' \supseteq S$ saturé tel que $S' \nvdash_{\mathbf{R}'_0} \bot$.*

Démonstration. Fixons une énumération sans répétition $\{P_0, P_1, \ldots\}$ de $\mathcal{P}_0$ dénombrable. On construit une séquence infinie d'ensembles de clauses $(S_n)_{n\in\mathbb{N}}$:

$$S_0 \stackrel{\text{def}}{=} S \qquad\qquad S_{n+1} \stackrel{\text{def}}{=} \begin{cases} S_n \cup \{\{P_n\}\} & \text{si } S_n, \{P_n\} \nvdash_{\mathbf{R}'_0} \bot \\ S_n \cup \{\{\neg P_n\}\} & \text{sinon} \end{cases}$$

et l'ensemble $S' \stackrel{\text{def}}{=} \bigcup_{n\in\mathbb{N}} S_n$. L'ensemble S' est saturé par définition.

Commençons par montrer par récurrence sur n que, pour tout n,

$$S_n \nvdash_{\mathbf{R}'_0} \bot\,. \tag{‡}$$

Pour le cas de base, $S_0 = S \nvdash_{\mathbf{R}'_0} \bot$ par hypothèse. Pour l'étape de récurrence au rang $n+1$, supposons par l'absurde que $S_{n+1} \vdash_{\mathbf{R}'_0} \bot$. Alors d'une part $S_{n+1} = S_n \cup \{\{\neg P_n\}\}$ et donc $S_n, \{\neg P_n\} \vdash \bot$, et d'autre part $S_n, \{P_n\} \vdash_{\mathbf{R}'_0} \bot$. Par le lemme 10.11 de déduction, on en déduirait $S_n \vdash_{\mathbf{R}'_0} P_n$ et $S_n \vdash_{\mathbf{R}'_0} \neg P_n$. Mais alors, on pourrait appliquer (res) et déduire $S_n \vdash_{\mathbf{R}'_0} \bot$, en contradiction avec l'hypothèse de récurrence.

Montrons enfin que $S' \nvdash_{\mathbf{R}'_0} \bot$. Par l'absurde, si $S' \vdash_{\mathbf{R}'_0} \bot$, alors il en existerait une dérivation finie qui ne ferait appel qu'à un sous-ensemble fini F de clauses de S' tel que $F \vdash_{\mathbf{R}'_0} \bot$. Mais alors il existerait $n \in \mathbb{N}$ tel que $F \subseteq S_n$ et donc tel que $S_n \vdash_{\mathbf{R}'_0} \bot$, en contradiction avec (‡). □

Théorème 10.10 (complétude réfutationnelle de la résolution complète)**.** *Soit S un ensemble de clauses. Si S est insatisfiable alors $S \vdash_{\mathbf{R}'_0} \bot$.*

Démonstration du théorème de complétude réfutationnelle de la résolution. On procède par contraposée : on suppose $S \nvdash_{\mathbf{R}'_0} \bot$ et on montre que S est satisfiable. Par le lemme 10.9, il existe S' saturé qui contient S et tel que $S' \nvdash_{\mathbf{R}'_0} \bot$. On définit alors l'interprétation I telle que $I \vDash P_n$ si et seulement si $\{P_n\} \in S'$. Montrons que $I \vDash S'$, et donc que $I \vDash S$ puisque $S \subseteq S'$.

Soit $C = \{Q_1, \ldots, Q_k, \neg R_1, \ldots, \neg R_\ell\}$ une clause de S'. Par l'absurde, si $I \nvDash C$, alors $I \vDash \neq Q_i$ pour tout $1 \leq i \leq k$ et $I \vDash R_j$ pour tout $1 \leq j \leq \ell$. Par définition de I, $\{\neg Q_i\} \in S'$ et $\{R_j\} \in S'$ pour tout $1 \leq i \leq k$ et $1 \leq j \leq \ell$. Par $k + \ell$ applications de (res), on obtiendrait alors une dérivation de $C, \{\neg Q_1\}, \ldots, \{\neg Q_k\}, \{R_1\}, \ldots, \{R_\ell\} \vdash_{\mathbf{R}'_0} \bot$, en contradiction avec $S' \nvdash_{\mathbf{R}'_0} \bot$. □

10.2. **Complétude propositionnelle.** Nous nous intéressons maintenant à la complétude du système de la figure 6 en entier. Le lemme suivant donne un pendant syntaxique au fait que $S \vDash \varphi \vee \psi$ si et seulement si $S \cup \{\neg\varphi\} \vDash \psi$.

[David et al., 2003, lem. 7.4.7], mais j'ai des doutes sur leur preuve. L'hypothèse $S \cup \{\overline{\ell}\} \vdash_{\mathbf{R}_0} C$ suffit pour le théorème de complétude propositionnelle.

Lemme 10.11 (déduction)**.** *Soit S un ensemble de clauses, C une clause, et ℓ un littéral. Si $S \cup \{\overline{\ell}\} \vdash_{\mathbf{R}'_0} C$ alors $S \vdash_{\mathbf{R}'_0} C, \ell$.*

Démonstration. Par induction structurelle sur la dérivation de $S \cup \{\overline{\ell}\} \vdash_{\mathbf{R}'_0} C$. Pour le cas de base où aucune règle n'a été appliquée,
- soit $C = \overline{\ell}$ et par la règle du tiers exclu (te) $\vdash_{\mathbf{R}'_0} C, \ell$,
- soit $C \in S$ et par la règle d'affaiblissement (aff) $S \vdash_{\mathbf{R}'_0} C, \ell$.

Pour l'étape d'induction, on fait une distinction de cas selon la dernière règle appliquée.

Pour (te) : alors $C = P, \neg P$ et par affaiblissement $\vdash_{\mathbf{R}'_0} C, \ell$.

Pour (aff) : alors $C = C', \ell'$ et $S \cup \{\overline{\ell}\} \vdash_{\mathbf{R}'_0} C'$. Par hypothèse d'induction $S \vdash_{\mathbf{R}'_0} C', \ell$ et par affaiblissement $S \vdash_{\mathbf{R}'_0} C', \ell', \ell$.

Pour (res) : alors $C = C_1, C_2$, $S \cup \{\overline{\ell}\} \vdash_{\mathbf{R}'_0} C_1, P$ et $S \cup \{\overline{\ell}\} \vdash_{\mathbf{R}'_0} \neg P, C_2$. Par hypothèse d'induction, $S \vdash_{\mathbf{R}'_0} C_1, P, \ell$ et $S \vdash_{\mathbf{R}'_0} \neg P, C_2, \ell$. Par résolution $S \vdash_{\mathbf{R}'_0} C_1, C_2, \ell$. □

Théorème 10.12 (complétude)**.** *Soit S un ensemble de clauses et C une clause. Si $S \models C$ alors $S \vdash_{\mathbf{R}'_0} C$.*

Démonstration. Écrivons $C = \{\ell_1, \ldots, \ell_k\}$. Si $S \models C$, alors $S \cup \{\overline{\ell_1}\} \cup \cdots \cup \{\overline{\ell_k}\}$ est insatisfiable. Par le théorème 10.6 de complétude réfutationnelle de la résolution, $S \cup \{\overline{\ell_1}\} \cup \cdots \cup \{\overline{\ell_k}\} \vdash_{\mathbf{R}_0} \bot$. Par k applications du lemme 10.11 de déduction, $S \vdash_{\mathbf{R}'_0} C$. □

11. Algorithme de Davis, Putnam, Logemann et Loveland

[Goubault-Larrecq et Mackie, 1997, sec. 2.4.3]

Couramment appelé *DPLL* d'après ses inventeurs, cet algorithme est au cœur des solveurs SAT actuels. C'est un raffinement d'un algorithme dû à Davis et Putnam, qui s'appuyait sur la résolution, mais DPLL n'utilise pas la résolution. L'algorithme DPLL 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. Nous allons en voir en section 11.2 une version élémentaire non déterministe, présentée sous la forme d'un système de réécriture. Mais tout d'abord, nous allons voir dans l'section 11.1 que la base de DPLL, qui est la recherche de modèle par simplification, n'est finalement pas si éloignée de la résolution.

11.1. Recherche de modèle par simplification.

Soit S un ensemble de clauses. On définit la *simplification* de S par un littéral ℓ comme l'ensemble de clauses

$$S[\top/\ell] \stackrel{\text{def}}{=} \{C \mid \big((C, \overline{\ell}) \in S \text{ ou } \overline{\ell} \notin C \in S\big) \text{ et } \ell \notin C\}$$

où l'on a éliminé les clauses de S contenant ℓ et simplifié les clauses de S de la forme $C, \overline{\ell}$ en leur enlevant $\overline{\ell}$; on peut remarquer que cela revient effectivement à substituer $\top$ à ℓ et $\bot$ à $\overline{\ell}$ dans toutes les clauses de S et à simplifier le résultat. Pour une interprétation I, on note $I[\top/\ell]$ pour l'interprétation qui associe $\top$ à P si $\ell = P$, $\bot$ à P si $\ell = \neg P$, et Q^I à toute proposition $Q \notin \mathcal{P}_0(\ell)$. On a alors par le lemme 3.1 de substitution propositionnelle

$$I \models S[\top/\ell] \qquad \text{si et seulement si} \qquad I[\top/\ell] \models S\,. \tag{$\star$}$$

Comme son nom l'indique, la *recherche par simplification* vise à trouver une interprétation qui satisfait toutes les clauses d'un ensemble S de clauses, et est composée de la seule règle de *scission* (split) de la figure 7.

$$\frac{S[\top/\neg P] \quad S[\top/P]}{S}\ \text{(split)}$$

Figure 7. Règle de la recherche de modèle par simplification.

La recherche de modèle *échoue* s'il existe une dérivation à l'aide de la règle (split) telle que tous les ensembles des feuilles contiennent la clause vide $\bot$; elle *réussit* s'il existe une branche qui termine sur l'ensemble vide $\emptyset$ de clauses.

Exemple 11.1. La recherche par simplification réussit pour l'ensemble de clauses $\{\neg P \vee \neg Q \vee P,\ \ Q \vee \neg R,\ \ \neg R\}$, comme le montre la dérivation suivante.

$$\dfrac{\dfrac{\dfrac{\bot \qquad \emptyset}{\emptyset \qquad \{Q\}}\,\text{(split)}}{\{Q, \neg R\},\ \{\neg R\}}\,\text{(split)} \qquad \dfrac{\dfrac{\bot \qquad \emptyset}{\emptyset \qquad \{Q\}}\,\text{(split)}}{\{Q, \neg R\},\ \{\neg R\}}\,\text{(split)}}{\{\neg P, \neg Q, P\},\ \{Q, \neg R\},\ \{\neg R\}}\,\text{(split)}$$

On voit sur cette dérivation que les interprétations partielles $[\bot/P, \bot/R]$, $[\bot/P, \top/R, \top/Q]$, $[\top/P, \bot/R]$ et $[\top/P, \top/R, \top/Q]$ correspondant aux quatre feuilles étiquetées par $\emptyset$ satisfont cet ensemble de clauses. Les deux interprétations partielles $[\bot/P, \top/R, \bot/Q]$ et $[\top/P, \top/R, \bot/Q]$ correspondant aux deux feuilles étiquetées par des ensembles contenant la clause $\bot$ ne satisfont pas l'ensemble $\{\neg P \vee \neg Q \vee P,\ \ Q \vee \neg R,\ \ \neg R\}$.

Exemple 11.2. Considérons à nouveau l'ensemble de clauses de l'exemple 2.3

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

La recherche par simplification échoue pour cet ensemble, comme le montre la dérivation suivante où toutes les feuilles sont étiquetées par un ensemble contenant $\bot$.

$$\dfrac{\dfrac{\dfrac{\bot \qquad \bot}{\{R\},\ \{\neg R\}}\,\text{(split)} \qquad \bot}{\{Q, R\},\ \{\neg Q\}}\,\text{(split)} \qquad \dfrac{\dfrac{\bot \qquad \bot}{\{Q\},\ \{\neg Q\}}\,\text{(split)} \qquad \bot}{\{Q, R\},\ \{\neg Q, R\},\ \{\neg R\}}\,\text{(split)}}{\{P, Q, R\},\ \{Q, R\},\ \{\neg P, \neg Q, R\},\ \{\neg P, \neg R\},\ \{P \vee \neg Q\}}\,\text{(split)}$$

Remarquons que la dérivation de l'exemple 11.2 est à nouveau isomorphe à l'arbre sémantique fermé de la figure 2 ainsi qu'à la dérivation de l'exemple 10.2. Ce n'est pas un hasard : même si l'on n'en fera la preuve, les dérivations de recherche par simplification qui échouent sont elles aussi isomorphes aux arbres sémantiques fermés et aux dérivations non redondantes en résolution. À noter que comme les branches en recherche par simplification sont de longueur au plus $|\mathcal{P}_0(S)|$, et la recherche de modèle par simplification fournit donc un algorithme en NP pour SAT.

11.2. **Branches de succès et algorithme DPLL.** L'algorithme DPLL est un raffinement de la recherche de modèle par simplification. Comme l'objectif est de trouver une branche de succès, qui termine sur l'ensemble vide $\emptyset$, nous allons en donner une formalisation en terme d'un système de réécriture sur les ensembles de clauses.

On définit l'ensemble des littéraux *purs* d'un ensemble F de clauses comme l'ensemble des littéraux ℓ tels que $\overline{\ell}$ n'apparaît dans aucune clause de F :

$$\text{Pure}(F) \stackrel{\text{def}}{=} \{\ell \mid \forall C \in F\,.\,\overline{\ell} \notin C\}\ .$$

Les règles de la figure 8 cherchent à montrer qu'un ensemble fini F est satisfiable en le réduisant à l'ensemble vide. Elles travaillent sur des ensembles F de clauses ; pour un tel ensemble F, une clause C et un littéral ℓ, « F, C » dénote l'ensemble $F \cup \{C\}$ et « $\{C, \ell\}$ » la clause $C \vee \ell$.

$$\begin{aligned}
F, \{P, \neg P, C\} &\to_{\text{dpll}} F && && \text{(taut)}\\
F, \{\ell\} &\to_{\text{dpll}} F[\top/\ell] && && \text{(unit)}\\
F &\to_{\text{dpll}} F[\top/\ell] && \text{où } \ell \in \text{Pure}(F) && \text{(pure)}\\
F &\to_{\text{dpll}} F[\top/P] && \text{où } P \in \mathcal{P}_0(F) && (\text{split}_P)\\
F &\to_{\text{dpll}} F[\top/\neg P] && \text{où } P \in \mathcal{P}_0(F) && (\text{split}_{\neg P})
\end{aligned}$$

FIGURE 8. Règles de réécriture de DPLL.

Exemple 11.3. La formule de l'exemple 11.1 $(\neg P \vee \neg Q \vee P) \wedge (Q \vee \neg R) \wedge (\neg R)$ est satisfiable puisque

$$\{\neg P, \neg Q, P\}, \{Q, \neg R\}, \{\neg R\} \xrightarrow{(\text{split}_P)}_{\text{dpll}} \{Q, \neg R\}, \{\neg R\} \xrightarrow{(\text{split}_{\neg R})}_{\text{dpll}} \emptyset$$

où les choix de propositions sont indiqués en orange : l'interprétation $[\top/P, \bot/R]$ satisfait en effet la formule.

11.2.1. *Correction et complétude.* Il est aisé de voir que F est satisfiable si et seulement si $F \to^*_{\text{dpll}} \emptyset$ à l'aide des règles (split_P) et $(\text{split}_{\neg P})$ pour $P \in \mathcal{P}_0(F)$: celles-ci cherchent une interprétation $I \in \mathbb{B}^{\mathcal{P}_0(F)}$ qui satisfait F.

Proposition 11.4. *Soit F un ensemble fini de clauses. Alors les règles de la figure 8 sont correctes et complètes : F est satisfiable si et seulement si $F \to^*_{\text{dpll}} \emptyset$.*

Démonstration. Pour la correction, c'est-à-dire pour montrer que $F \to^*_{\text{dpll}} \emptyset$ implique F satisfiable, il suffit d'observer que si F' est satisfiable – disons par une interprétation I telle que $I \vDash F'$ – et $F \to_{\text{dpll}} F'$ par une des règles de la figure 8, alors F est satisfiable :

- **Pour (taut) :** alors $F = F' \cup \{\neg P \vee P \vee C\}$; comme $I \vDash \neg P \vee P$ pour toute proposition P, $I \vDash \neg P \vee P \vee C$ pour toute clause C et donc $I \vDash F$.
- **Pour (unit) :** alors $F' = F''[\top/\ell]$ et $F = F'' \cup \{\ell\}$ pour une clause unitaire ℓ de F'' : on a $I[\top/\ell] \vDash F''$ par l'équation $(\star)$ et donc $I[\top/\ell] \vDash F$.
- **Pour (pure) :** alors $F' = F[\top/\ell]$ pour un littéral ℓ : on a $I[\top/\ell] \vDash F$ par l'équation $(\star)$.
- **Pour (split_P) :** alors $F' = F[\top/P]$ pour une proposition P : on a $I[\top/P] \vDash F$ par l'équation $(\star)$.
- **Pour ($\text{split}_{\neg P}$) :** alors $F' = F[\top/\neg P]$ pour une proposition P : on a $I[\bot/P] \vDash F$ par l'équation $(\star)$.

Pour la complétude, c'est-à-dire pour montrer que F satisfiable implique $F \to^*_{\text{dpll}} \emptyset$, supposons que $I \vDash F$. On ordonne $\mathcal{P}_0(F)$ de manière arbitraire comme $P_1 < \cdots < P_n$. 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 $I \vDash P_i$ et alors $F_i \stackrel{\text{def}}{=} F_{i-1}[\top/P_i]$ et comme $I = I[\top/P_i]$, $I \vDash F_i$ par l'équation $(\star)$;
- soit $(\text{split}_{\neg P_i})$ si $I \vDash \neg P_i$ et alors $F_i \stackrel{\text{def}}{=} F_{i-1}[\top/\neg P_i]$ et comme $I = I[\top/\neg P_i]$, $I \vDash F_i$ par l'équation $(\star)$.

Comme $\mathcal{P}_0(F_i) = \{P_{i+1}, \ldots, P_n\}$ pour tout $0 \leq i \leq n$, F_n est un ensemble de clauses sans propositions. De plus, il ne peut pas contenir la clause vide puisque $I \vDash F_n$. Donc $F_n = \emptyset$. □

11.2.2. *Algorithme DPLL.* L'intérêt des règles (taut), (unit) et (pure) est qu'elles sont *inversibles*.

Proposition 11.5 (inversibilité)**.** *Soit F un ensemble de clauses. Si $F \xrightarrow{(\text{taut})}_{\text{dpll}} F'$, $F \xrightarrow{(\text{unit})}_{\text{dpll}} F'$ ou $F \xrightarrow{(\text{pure})}_{\text{dpll}} F'$ pour un ensemble de clauses F satisfiable, alors F' est aussi satisfiable.*

Démonstration. Supposons que I est une interprétation telle que $I \vDash F$.

- **Pour (taut) :** alors $F = F' \cup \{P \vee \neg P \vee C\}$. Comme $F' \subseteq F$, $I \vDash F'$.
- **Pour (unit) :** alors $F = F'' \cup \{\ell\}$ et $F' = F''[\top/\ell]$ pour une clause unitaire ℓ de F. Comme $I \vDash \ell$, $I = I[\top/\ell] \vDash F''$ et par l'équation $(\star)$, $I \vDash F''[\top/\ell]$.
- **Pour (pure) :** alors $F' = F[\top/\ell]$ où ℓ est un littéral pur de $\text{Pure}(F)$. Comme dans ce cas $F[\top/\ell] \subsetneq F$, $I \vDash F'$. □

💬 *L'algorithme DPLL est implémenté sous forme déterministe 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 ».

Les règles (taut), (unit) et (pure) peuvent donc être appliquées arbitrairement. La stratégie usuelle de l'algorithme DPLL est d'appliquer les règles (taut), (unit) et (pure) en priorité dans cet ordre avant d'essayer (split_P) ou ($\text{split}_{\neg P}$), de manière à accélérer la recherche de preuve en éliminant des points de choix.

Procédure DPLL(F)
1 **tant que** $F \neq \emptyset$ **faire**
2 **si** $\bot \in F$ **alors retourner** *non satisfiable*
3 **sinon si** $F \xrightarrow{\text{(taut)}}_{\text{dpll}} F'$ **alors** $F := F'$
4 **sinon si** $F \xrightarrow{\text{(unit)}}_{\text{dpll}} F'$ **alors** $F := F'$
5 **sinon si** $F \xrightarrow{\text{(pure)}}_{\text{dpll}} F'$ **alors** $F := F'$
6 **sinon**
7 **choisir** $P \in \mathcal{P}_0(F)$ *(choix non déterministe)*
8 **choisir** $F := F[\top/P]$ **ou** $F := F[\top/\neg P]$
9 **retourner** *satisfiable*

Exemple 11.6. Pour la formule $(\neg P \vee \neg Q \vee P) \wedge (Q \vee \neg R) \wedge (\neg R)$ de l'exemple 11.3, l'algorithme DPLL effectue la réécriture

$$\{\neg P, \neg Q, P\}, \{Q, \neg R\}, \{\neg R\} \xrightarrow{\text{(taut)}}_{\text{dpll}} \{Q, \neg R\}, \{\neg R\} \xrightarrow{\text{(unit)}}_{\text{dpll}} \emptyset$$

qui est une réécriture qui n'introduit aucun point de choix.

Pour finir, on peut observer que les dérivations de DPLL sont de longueur linéaire. On définit pour cela la *taille* d'un ensemble F de clauses comme la somme des tailles de ses clauses : $\|F\| \stackrel{\text{def}}{=} \sum_{C \in F} |C|$. On observe alors que si $F \to_{\text{dpll}} F'$, alors $\|F\| > \|F'\|$. Par conséquent, DPLL travaille en temps non déterministe polynomial : c'est un algorithme dans NP.

12. Clauses de Horn

Une *clause de Horn* est une clause où au plus un littéral est positif. On appelle une clause de Horn sans littéral positif $\neg Q_1 \vee \cdots \vee \neg Q_k$ une clause *négative* et on l'écrit plutôt sous la forme $\bot \Leftarrow Q_1 \wedge \cdots \wedge Q_k$. On appelle une clause de Horn avec un littéral positif $P \vee \neg Q_1 \vee \cdots \vee \neg Q_k$ une clause *non négative* et on l'écrit plutôt sous la forme $P \Leftarrow Q_1 \wedge \cdots \wedge Q_k$; une telle clause non négative peut aussi être vue comme une règle de déduction $\frac{Q_1 \cdots Q_k}{P}$.

Les clauses de Horn ont plusieurs intérêts :
- leur problème de satisfiabilité est le problème P-complet par excellence, doté d'un algorithme en temps linéaire,
- un système de déduction sur un ensemble fini n'est jamais qu'un ensemble de clauses de Horn,
- c'est le format employé en programmation logique, par exemple par le langage Prolog, et en bases de données par le langage Datalog pour la leçon **932** « Fondements des bases de données relationnelles ».

12.1. Modèle minimal.

On voit les interprétations comme des ensembles dans $2^{\mathcal{P}_0}$ ordonnés par inclusion $\subseteq$, qui forme un treillis complet. Une propriété importante des clauses de Horn est qu'elles ont un *modèle minimal* pour l'inclusion.

Propriété 12.1 (modèle minimal)**.** *Soit S un ensemble de clauses de Horn. Si S est satisfiable, alors il a un modèle minimal.*

Nous allons voir deux preuves différentes de la propriété 12.1. Dans les deux cas, cette propriété sera une conséquence d'un théorème plus intéressant.

12.1.1. *Modèle minimal par intersection.* La première preuve de la propriété 12.1 s'appuie sur une caractérisation sémantique des ensembles de clauses de HORN.

Théorème 12.2 (HORN propositionnel)**.** *Soit S un ensemble de formules propositionnelles. Alors les énoncés suivants sont équivalents :* [KNUTH, 2008, sec. 7.1.1, thm. H]

(i) *S est logiquement équivalent à un ensemble de clauses de HORN,*

(ii) $\mathrm{Sat}(S)$ *est fermé par intersection arbitraire et*

(iii) $\mathrm{Sat}(S)$ *est fermé par intersection finie.*

Démonstration. Pour *(i)* $\Rightarrow$ *(ii)*, soit S un ensemble de clauses de HORN et soit $(I_j)_{j\in J}$ une famille d'interprétations telle que $J \neq \emptyset$ et $I_j \vDash S$ pour tout $j \in J$. Soit $I \stackrel{\text{def}}{=} \bigcap_{j\in J} I_j$ leur intersection. On montre que pour toute clause $C \in S$, $I \vDash C$. Soit il existe un littéral positif P dans C tel que $I_j \vDash P$ pour tout $j \in J$ et alors $I \vDash P$, soit il existe $j \in J$ et un littéral négatif $\neg Q$ dans C tel que $I_j \nvDash Q$ et alors $I \nvDash Q$; dans tous les cas $I \vDash C$.

L'implication *(ii)* $\Rightarrow$ *(iii)* est évidente.

Pour *(iii)* $\Rightarrow$ *(i)*, soit S un ensemble de formules propositionnelles ; par la propriété 6.1 on peut supposer que S soit un ensemble de clauses. Supposons que $\mathrm{Sat}(S)$ soit fermé par intersection.

Si $\mathrm{Sat}(S) = \emptyset$, alors S est équivalent à l'ensemble $\{\bot\}$ constitué de la seule clause vide, qui est bien un ensemble de clauses de HORN.

Sinon, on montre que l'on peut remplacer chaque clause $C = (P_1 \vee \cdots \vee P_n \vee \neg Q_1 \vee \cdots \vee \neg Q_k) \in S$ par une clause de HORN $C' \subseteq C$ telle que $\mathrm{Sat}((S \setminus \{C\}) \cup \{C'\}) = \mathrm{Sat}(S)$. Comme $C' \subseteq C$, on a déjà $\mathrm{Sat}((S \setminus \{C\}) \cup \{C'\}) \subseteq \mathrm{Sat}(S)$. Plusieurs cas sont possibles.

- Si $n \leq 1$, C est une clause de HORN et on pose $C' \stackrel{\text{def}}{=} C$.
- Sinon, supposons que pour toute interprétation $I \in \mathrm{Sat}(S)$, $I \vDash \neg Q_1 \vee \cdots \vee \neg Q_k$. Alors $C' \stackrel{\text{def}}{=} \neg Q_1 \vee \cdots \vee \neg Q_k$ convient.
- Sinon, supposons qu'il existe $1 \leq i \leq n$ tel que, pour toute interprétation $I \in \mathrm{Sat}(S)$ telle que $I \nvDash \neg Q_1 \vee \cdots \vee \neg Q_k$, on ait $I \vDash P_i$. Alors $C' \stackrel{\text{def}}{=} P_i \vee \neg Q_1 \vee \cdots \vee \neg Q_k$ convient.
- Sinon, pour tout $1 \leq i \leq n$, il existe une interprétation $I_i \in \mathrm{Sat}(S)$ telle que $I_i \nvDash \neg Q_1 \vee \cdots \vee \neg Q_k$ et $I_i \nvDash P_i$. Mais alors $\bigcap_{1\leq i\leq n} I_i \nvDash C$, une contradiction : ce dernier cas n'apparaît pas. □

Démonstration de la propriété 12.1. Soit S un ensemble de clauses de HORN. Si S est satisfiable, alors $\mathrm{Sat}(S)$ est non vide et $\bigcap_{I\in\mathrm{Sat}(S)} I$ est le modèle minimal de S. □

12.1.2. *Construction du modèle minimal.* Soit S un ensemble de clauses de HORN. Cet ensemble détermine une fonction $f_S \colon 2^{\mathcal{P}_0} \to 2^{\mathcal{P}_0}$ définie par

$$f_S(I) \stackrel{\text{def}}{=} \{P \in \mathcal{P}_0 \mid \exists (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S \,.\, \forall 1 \leq i \leq k \,.\, Q_i \in I\} \,.$$

On peut noter que f_S est monotone : $I \subseteq I'$ implique $f_S(I) \subseteq f_S(I')$.

Lemme 12.3. *Soit S un ensemble de clauses non négatives de HORN. Alors $I \vDash S$ si et seulement si $f_S(I) \subseteq I$.*

Démonstration. Si $I \vDash S$, alors pour toute clause $C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S$, si $I \vDash Q_i$ pour tout $1 \leq i \leq k$, alors $I \vDash P$. Donc $f_S(I) \subseteq I$.

Inversement, si $f_S(I) \subseteq I$, alors pour toute clause $C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S$, soit il existe $1 \leq i \leq k$ tel que $I \nvDash Q_i$ et donc $I \vDash C$, soit pour tout $1 \leq i \leq k$, $I \vDash Q_i$ et alors nécessairement $I \vDash P$ et donc $I \vDash C$. □

En itérant la fonction f_S à partir de l'ensemble vide, on définit une chaîne d'interprétations $\emptyset \subseteq f_S(\emptyset) \subseteq f_S^2(\emptyset) \subseteq \cdots$ dont la limite est $I_S \stackrel{\text{def}}{=} \bigcup_{n\in\mathbb{N}} f_S^n(\emptyset)$.

[Winskel, 1993, sec. 4.4]

Lemme 12.4. *Soit S un ensemble de clauses de Horn. Alors I_S est la plus petite interprétation telle que $f_S(I_S) \subseteq I_S$.*

Démonstration. Montrons tout d'abord que $f_S(I_S) \subseteq I_S$. Pour toute clause non négative $C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S$, si $I_S \vDash Q_i$ pour tout $1 \leq i \leq k$, alors pour chaque $1 \leq i \leq k$ il existe un indice n_i tel que $f_S^{n_i}(\emptyset) \vDash Q_i$. Dès lors, si l'on définit $n \stackrel{\text{def}}{=} \max_{1 \leq i \leq k} n_i$, $f_S^n(\emptyset) \vDash Q_i$ pour tout $1 \leq i \leq k$ et donc $f_S^{n+1}(\emptyset) \vDash P$, ce qui implique $I_S \vDash P$.

Soit maintenant I une interprétation telle que $f_S(I) \subseteq I$. On montre par récurrence sur n que $f_S^n(\emptyset) \subseteq I$ pour tout n, ce qui impliquera $I_S \subseteq I$. Pour le cas de base au rang $n = 0$, $\emptyset \subseteq I$. Pour l'étape de récurrence $n+1$, comme par hypothèse de récurrence au rang $n+1$, $f_S^n(\emptyset) \subseteq I$ et comme f_S est monotone, $f_S^{n+1}(\emptyset) \subseteq f_S(I) \subseteq I$. □

Nous déduisons maintenant une nouvelle preuve de la propriété 12.1 : le modèle minimal de S est l'interprétation I_S que nous avons construite.

*Le théorème 12.5 peut être appliqué pour définir la sémantique dénotationnelle du langage Imp dans la leçon **930** ; voir [Winskel, 1993, p. 60].*

Théorème 12.5. *Soit S un ensemble de clauses de Horn. Alors S est satisfiable si et seulement si $I_S \vDash S$, et dans ce cas I_S est le modèle minimal de S.*

Démonstration. Soit $S' \subseteq S$ le sous-ensemble des clauses non négatives de S. Cet ensemble est toujours satisfiable, par exemple par $I \stackrel{\text{def}}{=} \mathcal{P}_0$ tout entier. Par les lemmes 12.3 et 12.4, I_S est la plus petite interprétation telle que $I_S \vDash S'$.

Supposons $I_S \vDash S$. Alors S est satisfiable, et il reste à montrer que I_S est le modèle minimal de S. En effet, si $I \subseteq I_S$ est une interprétation telle que $I \vDash S$, alors $I \vDash S'$ et par les lemmes 12.3 et 12.4, $I_S \subseteq I$.

Supposons maintenant S satisfiable et montrons que $I_S \vDash S$. Tout modèle I de S est en particulier un modèle de S', donc $I_S \subseteq I$ puisque I_S est minimal. De plus, pour toute clause négative $C = (\bot \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S$, comme $I \vDash C$, il existe $1 \leq i \leq k$ tel que $I \nvDash Q_k$. Donc $I_S \nvDash Q_k$ et $I_S \vDash C$. □

Voici enfin une propriété bien utile quand on raisonne sur des clauses de Horn.

Propriété 12.6. *Soit S un ensemble de clauses de Horn et $P \in \mathcal{P}_0$ une proposition. Alors $P \in I_S$ si et seulement s'il existe une clause $(P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S$ telle que, pour tout $1 \leq i \leq k$, $Q_i \in I_S$.*

Démonstration. Par le lemme 12.4, $f_S(I_S) \subseteq I_S$ donc si $(P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S$ est une clause telle que, pour tout $1 \leq i \leq k$, $Q_i \in I_S$, alors $P \in I_S$.

Inversement, supposons que pour toute clause $(P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S$, il existe $1 \leq i \leq k$ tel que $Q_i \notin I_S$, c'est-à-dire tel que pour tout n, $Q_i \notin f_S^n(\emptyset)$. Montrons par récurrence sur n que $P \notin f_S^n(\emptyset)$ pour tout n, et que donc $P \notin I_S$. Pour le cas de base au rang 0, $P \notin \emptyset = f_S^0(\emptyset)$. Pour l'étape de récurrence au rang $n+1$, par hypothèse de récurrence $P \notin f_S^n(\emptyset)$. Comme pour toute clause $(P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in S$, il existe $1 \leq i \leq k$ tel que $Q_i \notin f_S^n(\emptyset)$, $P \notin f_S(f_S^n(\emptyset)) = f_S^{n+1}(\emptyset)$. □

12.2. **Complexité.** Le problème de satisfiabilité associé aux clauses de Horn est la restriction suivante de SAT :

Problème (HornSAT)**.**
instance : un ensemble fini F de clauses de Horn
question : F est-il satisfiable ?

Une manière naturelle de résoudre HornSAT est de calculer l'interprétation I_F définie dans la section 12.1.2 : alors par le théorème 12.5, F est satisfiable si et seulement si $I_F \vDash F$. Dans l'énoncé suivant, $\|F\| \stackrel{\text{def}}{=} \sum_{C \in S} |C|$.

Proposition 12.7. *Soit F un ensemble fini de clauses de Horn. Alors on peut calculer I_F en temps déterministe linéaire $O(\|F\| \cdot e(|\mathcal{P}_0(F)|))$ où $e(n)$ borne le coût des opérations de recherche et d'insertion dans des sous-ensembles à n éléments de $\mathcal{P}_0(F)$.*

Démonstration. Pour un ensemble F donné en entrée, on définit des ensembles $(I_n)_n$ par

$$I_0 \stackrel{\text{def}}{=} \emptyset\ , \qquad\qquad I_{n+1} \stackrel{\text{def}}{=} f_S(\bigcup_{j\leq n} I_j) \setminus \bigcup_{j\leq n} I_j\ .$$

Montrons par récurrence sur n que $f_S^n(\emptyset) = \bigcup_{j\leq n} I_j$. Pour le cas de base au rang $n = 0$, $I_0 = \emptyset$. Pour l'étape de récurrence au rang $n+1$, $\bigcup_{j\leq n+1} I_j = (f_S(\bigcup_{j\leq n} I_j) \setminus \bigcup_{j\leq n} I_j) \cup \bigcup_{j\leq n} I_j = f_S(\bigcup_{j\leq n} I_j) \cup \bigcup_{j\leq n} I_j \stackrel{\text{h.r}}{=} f_S(f_S^n(\emptyset)) \cup f_S^n(\emptyset) = f_S^{n+1}(\emptyset)$.

On peut noter que $I_n \subseteq \mathcal{P}_0(F)$ pour tout n et que les ensembles I_i et I_j sont disjoints pour tout $i \neq j$. De plus, si $I_{n+1} = \emptyset$ pour un certain n, alors $f_S(\bigcup_{j\leq n} I_j) \subseteq \bigcup_{j\leq n} I_j$ et donc $I_F = \bigcup_{0\leq j\leq n} I_n$; on a aussi $I_{n'} = \emptyset$ pour tout $n' > n$. Cela montre que $I_F = \bigcup_{0\leq j\leq n} I_n$ pour un certain $n \leq |\mathcal{P}_0(F)|$.

Il reste à transcrire ces idées en un algorithme travaillant en temps linéaire, qui va calculer les ensembles I_n jusqu'à ce que $I_{n+1} = \emptyset$ et retourner leur union. On suppose que les clauses non-négatives $P \Leftarrow Q_1 \wedge \cdots \wedge Q_k$ sont représentées comme des paires $(P, Q_1 \cdots Q_k)$ associant une proposition positive à une liste de propositions négatives.

[Knuth, 2008, sec. 7.1.1, algo. C], et [Harrison, 1978, p. 97], [Sippu et Soisalon-Soininen, 1988, thm. 4.14] pour des versions de cet algorithme sur les grammaires algébriques.

Fonction horn(F)
1 $n := 1$
2 $I_1 := \varepsilon$ *(Les $(I_n)_n$ sont représentés par des listes de propositions, initialement vides)*
3 $I := \emptyset$ *(L'interprétation I est représentée comme un ensemble de propositions, initialement vide)*
4 **pour chaque** $Q \in \mathcal{P}_0(F)$ **faire**
5 $\quad$ *contient*$_{\neg Q} := \varepsilon$ *(Liste des clauses contenant $\neg Q$, initialement vide)*
6 **pour chaque** $C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in F$ **faire**
7 $\quad$ *compte*$_C := k$ *(Nombre de littéraux à traiter de la clause C)*
8 $\quad$ **pour chaque** $1 \leq i \leq k$ **faire**
9 $\quad\quad$ *contient*$_{\neg Q_i} := C \cdot$ *contient*$_{\neg Q_i}$ *(ajout en tête de liste)*
10 $\quad$ **si** $k = 0$ **alors**
11 $\quad\quad$ $I := I \cup \{P\}$ *(insertion dans un ensemble)*
12 $\quad\quad$ $I_1 := P \cdot I_1$
(Boucle principale)
13 **tant que** $I_n \neq \varepsilon$ **faire**
14 $\quad$ $I_{n+1} := \varepsilon$
15 $\quad$ **pour chaque** $Q \in I_j$ **faire**
16 $\quad\quad$ **pour chaque** $C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in$ *contient*$_{\neg Q}$ **faire**
17 $\quad\quad\quad$ **si** $P \notin I$ **alors** *(recherche dans un ensemble)*
18 $\quad\quad\quad\quad$ *compte*$_C :=$ *compte*$_C - 1$
19 $\quad\quad\quad\quad$ **si** *compte*$_C = 0$ **alors** *(c'est-à-dire si $Q_i \in I$ pour tout $1 \leq i \leq k$)*
20 $\quad\quad\quad\quad\quad$ $I := I \cup \{P\}$
21 $\quad\quad\quad\quad\quad$ $I_{n+1} := P \cdot I_{n+1}$
22 $\quad$ $n := n + 1$
23 **retourner** I *($I = I_F$ le modèle minimal de F)*

Cet algorithme commence par initialiser deux structures aux lignes 4 à 12 :

- pour chaque proposition Q, la liste *contient*$_{\neg Q}$ des clauses $C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k)$ où Q apparaît négativement : C apparaît alors $\big|\{1 \leq i \leq k \mid Q_i = Q\}\big|$ fois dans *contient*$_{\neg Q}$,

— pour chaque clause $C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k)$, un compteur compte_C initialisé au nombre de propositions k de son corps.

À l'issue de l'initialisation, on a donc les invariants suivants, qui vont rester vrais à la ligne 13 après chaque tour de la boucle principale :

(1) les ensembles I_j pour $j \leq n$ ont été calculés,

(2) $I = \bigcup_{j \leq n} I_j$,

(3) $\forall C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in F$, $\mathit{compte}_C = \big|\{1 \leq i \leq k \mid Q_i \notin \bigcup_{j<n} I_j\}\big|$.

La mise à jour de I_{n+1} à la ligne 19 se fait en effet quand $\mathit{compte}_C = 0$, c'est-à-dire quand toutes les propositions Q_i sont dans I. L'algorithme termine quand $I_{n+1} = \emptyset$ et alors $I_F = I = \bigcup_{j \leq n} I_j$.

En ce qui concerne la complexité de l'algorithme, la phase d'initialisation est clairement en temps $O(\|F\| \cdot e(|\mathcal{P}_0(F)|))$. Pour la boucle principale, il convient de remarquer que, comme les ensembles I_j sont mutuellement disjoints, on ne passe par la ligne 15 qu'au plus une fois par proposition $Q \in \mathcal{P}_0(F)$ pendant l'exécution. Cela signifie que pour chaque clause $C = (P \Leftarrow Q_1 \wedge \cdots \wedge Q_k) \in F$, on ne passe par la ligne 16 qu'au plus k fois sur la totalité de l'exécution de l'algorithme ; la boucle principale s'exécute donc elle aussi en $O(\|F\| \cdot e(|\mathcal{P}_0(F)|))$. □

Corollaire 12.8. HornSAT *est en temps déterministe linéaire sur une machine RAM.*

Démonstration. Les opérations de recherche et d'insertion sur $2^{\mathcal{P}_0(F)}$ sont en temps constant $O(1)$ en utilisant des vecteurs de bits de longueur $|\mathcal{P}_0(F)|$. Par le théorème 12.5, on peut utiliser l'algorithme de la proposition 12.7 pour calculer I_F en temps linéaire et vérifier dans une deuxième phase que $I_F \vDash C$ pour toutes les clauses négatives $C \in F$, ce qui s'effectue aussi en temps linéaire. Alternativement, on aurait pu traiter $\bot$ comme une proposition et vérifier si $\bot \notin I_F$ dans le résultat de l'algorithme. □

Théorème 12.9. HornSAT *est* P*-complet.*

Démonstration. Par le corollaire 12.8, HornSAT est dans P. Il reste donc à montrer qu'il est P-difficile. On va réduire pour cela depuis le problème VALEUR-CIRCUIT-MONOTONE suivant.

Problème (VALEUR-CIRCUIT-MONOTONE).
instance : γ un circuit monotone
question : $[\![\gamma]\!] = \top$?

Dans ce problème, un *circuit monotone* est un circuit au sens de la section 1 pour la syntaxe abstraite

$$\gamma ::= \bot \mid \top \mid \gamma \vee \gamma \mid \gamma \wedge \gamma \,. \qquad \text{(circuits monotones)}$$

On peut voir que la sémantique $[\![\gamma]\!]$ de γ ne dépend pas de l'interprétation, c'est un élément de $\mathbb{B}$:

$$[\![\bot]\!] = \bot \,, \qquad [\![\top]\!] = \top \,, \qquad [\![\gamma \vee \gamma']\!] = [\![\gamma]\!] \vee [\![\gamma']\!] \,, \qquad [\![\gamma \wedge \gamma']\!] = [\![\gamma]\!] \wedge [\![\gamma']\!] \,.$$

[Papadimitriou, 1993, pp. 170–171]. On a en effet une réduction en espace logarithmique depuis VALEUR-CIRCUIT *(c.f. [Perifel, 2014, prop. 5-AB] et [Arora et Barak, 2009, thm. 6.30]), qui est le même problème pour des circuits où l'opération $\neg$ est aussi permise : la réduction fait simplement $|\gamma|$ passes successives sur le circuit et applique une étape de la mise sous forme normale négative à chaque passe.*

VALEUR-CIRCUIT-MONOTONE est un problème P-complet classique.

Soit $\langle\gamma\rangle$ une instance de VALEUR-CIRCUIT-MONOTONE. On utilise une proposition distincte $P_{\gamma'}$ pour chaque porte γ' de γ. On construit un ensemble fini F de clauses de Horn non négatives

$$F \stackrel{\text{def}}{=} \bigcup_{\gamma' \text{ porte de } \gamma} F_{\gamma'}$$

où l'on définit un ensemble fini $F_{\gamma'}$ de clauses de Horn non négatives pour chaque porte γ' par

$$F_\bot \stackrel{\text{def}}{=} \emptyset \,, \qquad F_\top \stackrel{\text{def}}{=} \{P_\top \Leftarrow \top\} \,,$$
$$F_{\gamma_1 \vee \gamma_2} \stackrel{\text{def}}{=} \{P_{\gamma_1 \vee \gamma_2} \Leftarrow P_{\gamma_1}, P_{\gamma_1 \vee \gamma_2} \Leftarrow P_{\gamma_2}\} \,, \qquad F_{\gamma_1 \wedge \gamma_2} \stackrel{\text{def}}{=} \{P_{\gamma_1 \vee \gamma_2} \Leftarrow P_{\gamma_1} \wedge P_{\gamma_2}\} \,.$$

Montrons par induction structurelle sur les portes γ' que $P_{\gamma'} \in I_F$ le modèle minimal de F si et seulement si $[\![\gamma']\!] = \top$. On utilise dans chaque cas la propriété 12.6 :

Pour le cas de base $\gamma' = \bot$: alors $P_\bot \notin I_F$ et $[\![\bot]\!] = \bot$.

Pour le cas de base $\gamma' = \top$: alors $P_\top \in I_F$ et $[\![\top]\!] = \top$.

Pour l'étape d'induction $\gamma' = \gamma_1 \vee \gamma_2$: alors $P_{\gamma_1 \vee \gamma_2} \in I_F$ si et seulement si $P_{\gamma_1} \in I_F$ ou $P_{\gamma_2} \in I_F$, ce qui par hypothèse d'induction est si et seulement si $[\![\gamma_1]\!] = \top$ ou $[\![\gamma_2]\!] = \top$, ce qui par définition de $[\![\gamma_1 \vee \gamma_2]\!]$ est si et seulement si $[\![\gamma_1 \vee \gamma_2]\!] = \top$.

Pour l'étape d'induction $\gamma' = \gamma_1 \wedge \gamma_2$: alors $P_{\gamma_1 \wedge \gamma_2} \in I_F$ si et seulement si $P_{\gamma_1} \in I_F$ et $P_{\gamma_2} \in I_F$, ce qui par hypothèse d'induction est si et seulement si $[\![\gamma_1]\!] = \top$ et $[\![\gamma_2]\!] = \top$, ce qui par définition de $[\![\gamma_1 \wedge \gamma_2]\!]$ est si et seulement si $[\![\gamma_1 \wedge \gamma_2]\!] = \top$.

Pour conclure, le circuit γ s'évalue à $\top$ si et seulement si $P_\gamma \in I_F$, si et seulement si l'ensemble de clauses de Horn $F \cup \{\bot \Leftarrow P_\gamma\}$ est insatisfiable. Ce dernier ensemble de clauses peut être construit en espace logarithmique, donc nous venons d'exhiber une réduction logarithmique de VALEUR-CIRCUIT-MONOTONE à ¬HornSAT : HornSAT est difficile pour coP = P. □

Il s'agit d'une réduction many-one *:* VALEUR-CIRCUIT-MONOTONE $\leq^m_{\mathsf{L}}$ ¬HornSAT.

12.3. **Application à la preuve par saturation en résolution.** Les algorithmes cherchant à déterminer si la clause vide peut être dérivée travaillent habituellement par *saturation* : on calcule l'ensemble des clauses dérivables, c'est-à-dire l'ensemble des clauses C telles que $F \vdash_{\mathbf{R}_0} C$, et on vérifie si $\bot$ apparaît dans cet ensemble. Le problème de décision visé est le suivant.

Problème (RÉFUTATION)**.**
instance : Un ensemble fini F de clauses.
question : Est-ce que $F \vdash_{\mathbf{R}_0} \bot$?

Par la correction et la complétude réfutationnelle de la résolution de la résolution, et comme VALIDITÉ *est* coNP*-complet,* RÉFUTATION *est* coNP*-complet. Le but ici est de donner un algorithme déterministe « efficace » pour ce problème.*

Tout système de déduction peut être vu comme un ensemble de clauses de Horn. Le calcul de l'ensemble des clauses dérivables par résolution depuis un ensemble fini F de clauses est alors exactement celui du modèle minimal de l'ensemble de clauses de Horn correspondant à F.

Corollaire 12.10. *L'algorithme de saturation basé sur la proposition 12.7 résout* RÉFUTATION *en temps déterministe* $|F| \cdot 2^{O(|\mathcal{P}_0(F)|)}$.

C'est aussi la complexité de l'algorithme de recherche exhaustive pour SAT *qui essaie tour à tour toutes les* $2^{|\mathcal{P}_0(F)|}$ *interprétations possibles sur chacune des* $|F|$ *clauses.*

Démonstration. Étant donné F un ensemble fini de clauses sur un ensemble fini $L \stackrel{\text{def}}{=} \{\ell \mid \exists C \in F . \ell \in C\}$ de littéraux, on construit un ensemble de « méta-clauses » de Horn F' sur l'ensemble des « méta-propositions » 2^L : une méta-proposition est une clause construite sur L. On définit pour cela F' comme un ensemble groupant deux types de méta-clauses :

$$F' \stackrel{\text{def}}{=} \{C \Leftarrow \top \mid C \in F\} \cup \{(C, D) \Leftarrow (C, P) \wedge (\neg P, D) \mid C, D \in 2^L \text{ et } P, \neg P \in L\}$$

Alors $F \vdash_{\mathbf{R}_0} C$ si et seulement si $C \in I_{F'}$, et cela vaut en particulier pour la clause vide $\bot$. Enfin, la taille $\|F'\|$ est de $|F|$ pour les méta-clauses du premier type, et d'au plus $2^{2|L|} \cdot |\mathcal{P}_0(F)|$ méta-clauses du second type, où $|L| \leq 2|\mathcal{P}_0(F)|$. On représente les clauses comme des vecteurs de bits de longueur $|L|$; les opérations ensemblistes sur 2^{2^L} sont en $O(2^{|L|})$ en utilisant par exemple des arbres binaires de recherche équilibrés. La complexité annoncée découle donc de la proposition 12.7. □

Les algorithmes par saturation employés en pratique raffinent cet algorithme par différentes stratégies ; voir par exemple [Goubault-Larrecq et Mackie, 1997, sec. 7.4.3–7.4.6].

13. 2SAT

Rappelons que 2SAT est le problème kSAT pour $k = 2$:

Problème (2SAT)**.**
instance : un ensemble fini F de 2-clauses
question : F est-il satisfiable ?

Ce problème a une complexité considérablement plus simple que celle de SAT.

Théorème 13.1. 2SAT *est* NL*-complet.*

[Carton, 2008, thm. 4.47]

Démonstration de NL*-difficulté.* On exhibe une réduction en espace logarithmique depuis le problème NL-complet d'*accessibilité* dans les graphes dirigés finis.

*[Perifel, 2014, prop. 4AL], [Arora et Barak, 2009, thm. 4.18] pour le problème d'*ACCESSIBILITÉ.

Problème (ACCESSIBILITÉ)**.**
instance : un graphe dirigé fini (V, E) et deux sommets $s, t \in V$.
question : t est-il accessible depuis s ?

Soit $\langle (V, E), s, t\rangle$ une instance du problème d'accessibilité. On considère chaque sommet de V comme une proposition. Soit l'ensemble fini de clauses $F_E \stackrel{\text{def}}{=} \{\neg v \vee v' \mid (v, v') \in E\}$. Montrons que $F_E \vDash \neg v \vee v'$ si et seulement si $v\ E^*\ v'$.

- On montre par récurrence sur n que $v\ E^n\ v'$ implique $F_E \vDash \neg v \vee v'$. Pour le cas de base, $v = v'$ et donc $\vDash \neg v \vee v$. Pour l'étape de récurrence, soit $v\ E^n\ v''\ E\ v'$; par hypothèse de récurrence, $F_E \vDash \neg v \vee v''$ et de plus $\neg v'' \vee v' \in F_E$, donc par une application de (res), $F_E \vDash \neg v \vee v'$.
- Inversement, voici une preuve syntaxique. Supposons que $F_E \vDash \neg v \vee v'$. Par complétude de la résolution propositionnelle, $F_E \vdash_{\mathbf{R}'_0} \neg v, v'$ et on montre par induction sur la dérivation correspondante que $v\ E^*\ v'$. Commençons par observer que toutes les clauses dérivables depuis F_E ont au moins deux littéraux ; comme la clause $\neg v, v'$ est dérivée, cela implique en particulier que la règle (aff) n'a pas été appliquée dans cette dérivation. Donc toutes les clauses dans cette dérivation de $F_E \vdash_{\mathbf{R}'_0} \neg v, v'$ ont exactement deux littéraux, un positif et un négatif.

  Pour le cas de base, la clause $\neg v, v'$ appartient déjà à F_E et alors $v\ E\ v'$. Pour l'étape d'induction, on fait une distinction selon la dernière règle employée :

  (1) Cas de (res) : on a $F_E \vdash_{\mathbf{R}'_0} \neg v, v''$ et $F_E \vdash_{\mathbf{R}'_0} \neg v'', v'$, donc par hypothèse d'induction $v\ E^*\ v''\ E^*\ v'$.

  (2) Cas de (te) : on a $v = v'$ donc $v\ E^*\ v'$.

Il s'agit d'une réduction many-one *:* ACCESSIBILITÉ $\leq^m_{\mathsf{L}}$ ¬2SAT.

Pour conclure, $s E^* t$ si et seulement si $F_E \vDash \neg s \vee t$, si et seulement si $F_E \cup \{s\} \cup \{\neg t\}$ n'est pas satisfiable ; cette instance de 2SAT est clairement constructible en espace logarithmique. Comme NL = coNL, cela montre que 2SAT est NL-difficile. □

[Carton, 2008, prop. 4.43], [Papadimitriou, 1993, pp. 184–187], [Knuth, 2008, sec. 7.1.1, thm. K]

Démonstration des bornes supérieures. Soit F un ensemble fini de 2-clauses vues comme des ensembles de littéraux. Si F contient la clause vide, il est insatisfiable et cela peut être détecté en espace logarithmique. Quitte à remplacer les clauses $C = \ell$ de F avec un seul littéral par deux clauses $\{\ell, P\}$ et $\{\ell, \neg P\}$ pour une proposition fraîche $P \notin \mathcal{P}_0(F)$, on peut donc supposer que chaque clause de F contient exactement deux littéraux distincts.

On construit en espace logarithmique le graphe fini orienté (V_F, E_F) avec pour ensemble de sommets $V_F \stackrel{\text{def}}{=} \{P, \neg P \mid P \in \mathcal{P}_0(F)\}$ et pour ensemble d'arcs $E_F \stackrel{\text{def}}{=} \{(\ell, \ell') \mid \{\overline{\ell}, \ell'\} \in F\}$; la figure 9 donne un exemple de cette construction. On montre que F est insatisfiable si et seulement s'il existe une proposition P et un chemin de P à $\neg P$ et de $\neg P$ à P dans (V_F, E_F) ; cette dernière propriété est bien vérifiable en NL.

Il s'agit plus exactement d'une réduction de Turing *: la preuve montre que* ¬2SAT $\leq^T_{\mathsf{L}}$ ACCESSIBILITÉ *et donc que* 2SAT $\in$ coNL = NL.

Supposons qu'une telle proposition P existe mais qu'il existe une interprétation I qui satisfasse F. Supposons $I \vDash P$ (le cas où $I \vDash \neg P$ est symétrique). Montrons par récurrence sur n que pour tout ℓ tel que $P\ E^n_F\ \ell$, $I \vDash \ell$. C'est vrai pour le cas de base $\ell = P$. Pour l'étape de récurrence, si $P\ E^n_F\ \ell'\ E_F\ \ell$, alors par hypothèse de récurrence, $I \vDash \ell'$; de plus $\{\overline{\ell'}, \ell\} \in F$ et donc $I \vDash \ell$. On en déduit que $I \vDash \neg P$, contradiction.

Inversement, supposons qu'il n'existe pas de telle proposition. On montre par récurrence sur $|F|$ le nombre de clauses de F qu'il existe une interprétation I_F telle que $I_F \vDash F$. Pour le cas de base, $F = \emptyset$ est valide. Pour l'étape de récurrence pour F non vide, soit ℓ un littéral de

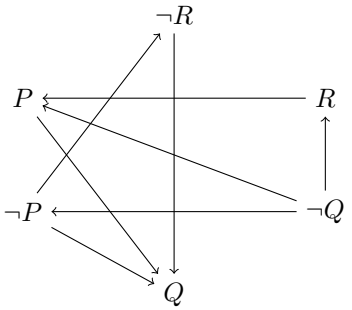

FIGURE 9. Le graphe (V_F, E_F) pour $F = \{\{P, Q\}, \{P, \neg R\}, \{\neg P, Q\}, \{Q, R\}\}$.

V_F. On définit les ensembles de littéraux $V_\ell \stackrel{\text{def}}{=} \{\ell' \mid \ell \; E_F^* \; \ell'\}$ et $\overline{V_\ell} \stackrel{\text{def}}{=} \{\overline{\ell'} \mid \overline{\ell'} \; E_F^* \; \overline{\ell}\}$; on peut noter que $\ell' \in V_\ell$ si et seulement si $\overline{\ell'} \in \overline{V_\ell}$.

Soit $F' \subseteq F$ le sous-ensemble de clauses de F qui n'utilise pas les littéraux de $V_\ell \cup \overline{V_\ell}$:

$$F' \stackrel{\text{def}}{=} \{C \in F \mid C \cap V_\ell = \emptyset \text{ et } C \cap \overline{V_\ell} = \emptyset\} \; .$$

Alors $(V_{F'}, E_{F'})$ est un sous-graphe de (V_F, E_F), donc il n'existe pas de proposition P avec un chemin de P à $\neg P$ et de $\neg P$ à P dans $(V_{F'}, E_{F'})$. Comme $|F'| < |F|$, par hypothèse de récurrence il existe une interprétation $I_{F'}$ telle que $I_{F'} \vDash F'$. Par exemple, dans la figure 9, si on a sélectionné $\ell = \neg R$, alors $V_{\neg R} = \{\neg R, Q\}$, $\overline{V_{\neg R}} = \{R, \neg Q\}$, $F' = \emptyset$, $V_{F'} = \{P, \neg P\}$ et $E_{F'} = \emptyset$.

On redéfinit l'interprétation $I_{F'}$ sur les littéraux de V_ℓ : on le fait de telle sorte que $I_F \vDash \ell'$ pour tout $\ell' \in V_\ell$ et donc que $I_F \nvDash \overline{\ell'}$ pour tout $\overline{\ell'} \in \overline{V_\ell}$; c'est possible car si $\ell' \in V_\ell$ alors $\overline{\ell'} \notin V_\ell$, puisque sinon on aurait un chemin $\ell \; E_F^* \; \overline{\ell'} \; E_F^* \; \overline{\ell}$. Pour les propositions $P \notin \mathcal{P}_0(V_\ell)$ on garde $P^{I_F} \stackrel{\text{def}}{=} P^{I_{F'}}$. Il reste à vérifier que $I_F \vDash C$ pour toutes les clauses $C \in S$.

- Si $C \notin F'$, alors deux cas sont possibles : $C \cap V_\ell \neq \emptyset$ ou $C \cap \overline{V_\ell} \neq \emptyset$. On montre qu'il existe dans tous les cas un littéral $\ell' \in C \cap V_\ell$, et donc que $I_F \vDash C$ puisque $I_F \vDash \ell'$. En effet, c'est évident dans le premier cas, et dans le second, s'il existe $\overline{\ell'} \in C \cap \overline{V_\ell}$, alors comme C contient deux littéraux distincts, $C = \{\overline{\ell'}, \ell''\}$ pour un certain littéral ℓ'' : on a donc $\ell' \; E_F \; \ell''$ et donc $\ell'' \in V_\ell$.
- Si $C \in F'$, alors $I_{F'} \vDash C$. Comme $I_F \cap \mathcal{P}_0(C) = I_{F'} \cap \mathcal{P}_0(C)$, par la propriété 2.1 on conclut que $I_F \vDash C$. □

Le théorème 13.1 montre que la satisfiabilité d'une formule en 2-CNF peut être résolue de manière très efficace. Tous les problèmes sur ces formules ne sont pas faciles pour autant.

Problème (Max$k$SAT)**.**
instance : Un ensemble fini F de k-clauses et un entier n.
question : Existe-t'il un sous-ensemble $F' \subseteq F$ avec $|F'| = n$ qui soit satisfiable ?

Théorème 13.2. Max$k$SAT *est* NP*-complet pour tout* $k \geq 2$.

Démonstration de l'appartenance à NP. On devine non déterministiquement le sous-ensemble F' de cardinal n et une interprétation partielle $I \in \mathbb{B}^{\mathcal{P}_0(F')}$ qui le satisfait ; ces objets sont de taille polynomiale en la taille de la représentation de F, et $I \vDash F'$ peut être vérifié en temps polynomial. □

Démonstration de NP*-difficulté.* Si $k \geq 3$, on a une réduction triviale de kSAT à Max$k$SAT en posant $n = |F|$. Il reste donc à montrer que Max2SAT est NP-difficile. On va exhiber pour cela une réduction en espace logarithmique depuis 3SAT. ∎ *[PAPADIMITRIOU, 1993, thm. 9.2]*

Soit une instance $\langle F \rangle$ de 3SAT, où F est un ensemble de 3-clauses. Soit p le nombre de clauses de F composées d'exactement trois littéraux ; les $|F| - p$ clauses restantes sont donc des 2-clauses et ne posent pas problème. Pour chacune des p clauses $C = \ell_1 \vee \ell_2 \vee \ell_3$ de F, on va choisir une proposition fraîche $P_C \notin \mathcal{P}_0(F)$ et remplacer C par l'ensemble F_C de dix 2-clauses suivant :

$$F_C \stackrel{\text{def}}{=} \{\ell_1,\ \ell_2,\ \ell_3,\ P_C,\ \overline{\ell_1} \vee \overline{\ell_2},\ \overline{\ell_2} \vee \overline{\ell_3},\ \overline{\ell_1} \vee \overline{\ell_3},\ \ell_1 \vee \neg P_C,\ \ell_2 \vee \neg P_C,\ \ell_3 \vee \neg P_C\}\ .$$

Montrons que, pour toute interprétation I, d'une part I satisfait au plus sept clauses de F_C, et d'autre part $I \vDash C$ si et seulement s'il existe un sous-ensemble $F'_C \subseteq F_C$ avec $|F'_C| = 7$ et une valeur de vérité $b \in \mathbb{B}$ tels que $I[b/P_C] \vDash F'_C$.

Si $\boldsymbol{I \vDash C}$: alors aux symétries entre ℓ_1, ℓ_2, et ℓ_3 près il y a trois cas possibles :

- **$\boldsymbol{I \vDash \ell_1, I \vDash \ell_2}$ et $\boldsymbol{I \vDash \ell_3}$:** alors sept clauses sont satisfaites par $I[\top/P_C]$ et six le sont si $I \nvDash P_C$.
- **$\boldsymbol{I \vDash \ell_1, I \vDash \ell_2}$ et $\boldsymbol{I \nvDash \ell_3}$:** alors sept clauses sont satisfaites par $I[\top/P_C]$ et sept le sont si $I \nvDash P_C$.
- **$\boldsymbol{I \vDash \ell_1, I \nvDash \ell_2}$ et $\boldsymbol{I \nvDash \ell_3}$:** alors sept clauses sont satisfaites par $I[\bot/P_C]$ et six le sont si $I \vDash P_C$.

Si $\boldsymbol{I \nvDash C}$: alors il n'y a qu'un cas possible :

- **$\boldsymbol{I \nvDash \ell_1, I \nvDash \ell_2}$ et $\boldsymbol{I \nvDash \ell_3}$:** alors six clauses sont satisfaites quelle que soit la valeur de vérité P_C^I.

Pour conclure, le nouvel ensemble de $|F| - p + 10p$ clauses que nous avons construit a un sous-ensemble satisfiable de $n \stackrel{\text{def}}{=} |F| - p + 7p$ clauses si et seulement si F était satisfiable. □

Annexe.

⚠ *Le contenu de cette annexe est probablement hors-programme, et est surtout justifié par ma curiosité personnelle, en particulier par rapport à la question suivante : peut-on montrer que* RÉFUTATION *est dans* coNP *par des techniques (essentiellement) syntaxiques ?*

Annexe A. Recherche de réfutation en résolution propositionnelle

Comme dans le cas du calcul des séquents, une *recherche de réfutation* en résolution propositionnelle pour un ensemble de clauses S procède de la racine $\bot$ vers les feuilles dans étiquetées par des clauses de S et vise à résoudre RÉFUTATION. Cependant, les règles de la résolution propositionnelle se prêtent assez mal à ce type d'algorithme, car à chaque application de la règle (res), il y a un choix à faire sur comment séparer la clause C, D de la conclusion en deux clauses C et D des prémisses. Nous allons voir que l'on peut modifier le système de règles pour éliminer ces choix.

A.1. **Système de recherche de réfutation.** Comme vu dans la section 10.1, les dérivations en résolution propositionnelle sont intimement liées aux arbres sémantiques. Une observation à faire sur la figure 2 est que l'on peut voir les branches comme « accumulant » des littéraux, jusqu'à ce qu'elles finissent par un nœud d'échec falsifié par cette accumulation. Par exemple, le quatrième nœud d'échec de la figure 2, étiqueté par $Q \vee R$, correspond aux résolvants $\neg P, R, Q$ de la dérivation de l'exemple 10.2 et à l'interprétation $[\top/P, \bot/R, \bot/Q]$. Ce point de vue suggère de légères modifications des règles de la résolution propositionnelle de la figure 6. La figure 10 présente un nouveau système de déduction à cette fin. On écrira $S \vdash_{\mathbf{Rf}} C$ s'il existe une dérivation de la clause C dans le système de la figure 10.

$$\frac{C, P \quad \neg P, C}{C}\ (\mathsf{res}') \qquad\qquad \frac{}{C}\ (\mathsf{sub}) \quad \text{où } \exists D \in S \,.\, D \subseteq C$$

Figure 10. Règles de la recherche de réfutation pour un ensemble S de clauses.

On peut commencer par observer que la règle (aff) est admissible dans ce nouveau système.

Lemme A.1 (affaiblissement)**.** *Soit S un ensemble de clauses, C une clause et ℓ un littéral. Si $S \vdash_{\mathbf{Rf}} C$, alors $S \vdash_{\mathbf{Rf}} C, \ell$.*

Démonstration. Par induction structurelle sur les dérivations. □

Comme dans le cas du calcul des séquents, les règles de la figure 10 sont inversibles : c'est une conséquence immédiate du lemme A.1 d'affaiblissement.

Corollaire A.2 (inversibilité)**.** *Les règles de recherche de réfutation sont inversibles.*

Ce nouveau système de déduction est équivalent à l'utilisation des deux règles (res) et (aff). Il est donc correct et réfutationnellement complet.

Proposition A.3. *Soit S un ensemble de clauses. Alors $S \vdash_{\mathbf{R}_0} \bot$ si et seulement si $S \vdash_{\mathbf{Rf}} \bot$.*

Démonstration. Commençons par montrer par induction structurelle sur une dérivation de $S \vdash_{\mathbf{Rf}} C$ pour une clause C que $S \vdash_{\mathbf{R}'_0} C$. Pour le cas de base, C vient d'être dérivée par (sub) pour une clause $D \in S$ telle que $D \subseteq C$, et on a donc une dérivation de C à partir de S utilisant $|C \setminus D|$ fois la règle (aff). Pour l'étape d'induction, C vient d'être dérivée par (res′) : il existe $P \in \mathcal{P}_0$ et deux dérivations de C, P et de $\neg P, C$ pour $S \vdash_{\mathbf{Rf}} C, P$ et $S \vdash_{\mathbf{Rf}} \neg P, C$ respectivement. Par hypothèse d'induction, il existe des dérivations de $S \vdash_{\mathbf{R}'_0} C, P$ et $S \vdash_{\mathbf{R}'_0} \neg P, C$. Par une application de (res), on a bien $S \vdash_{\mathbf{R}'_0} C$.

Dès lors, si $S \vdash_{\mathbf{Rf}} \bot$, alors $S \vdash_{\mathbf{R}'_0} \bot$. Par le théorème 10.3 de correction, S est insatisfiable. Par le théorème 10.6 de complétude réfutationnelle de la résolution, $S \vdash_{\mathbf{R}_0} \bot$.

Inversement, montrons par induction structurelle sur une dérivation de $S \vdash_{\mathbf{R}_0} C$ que $S \vdash_{\mathbf{Rf}} C$. Pour le cas de base, $C \in S$ et donc $S \vdash_{\mathbf{Rf}} C$ par (sub). Pour l'étape d'induction, on vient d'appliquer (res) sur C, P et $\neg P, D$ avec $S \vdash_{\mathbf{R}_0} C, P$ et $S \vdash_{\mathbf{R}_0} \neg P, D$. Par hypothèse d'induction, $S \vdash_{\mathbf{Rf}} C, P$ et $S \vdash_{\mathbf{Rf}} \neg P, D$. Par le lemme A.1 d'affaiblissement, $S \vdash_{\mathbf{Rf}} C, D, P$ et $S \vdash_{\mathbf{Rf}} \neg P, C, D$. Par une application de (res′), $S \vdash_{\mathbf{Rf}} C, D$.

On en déduit qu'en particulier $S \vdash_{\mathbf{R}_0} \bot$ implique $S \vdash_{\mathbf{Rf}} \bot$. □

A.2. **Recherche de réfutation.** On appelle *recherche de réfutation* le processus qui, partant de la clause vide, tente de construire une dérivation dans le système de la figure 10. Une *branche de réfutation* est une séquence potentiellement infinie de clauses $C_0, C_1, \dots$ avec $C_0 = \bot$, calculée par la recherche d'une réfutation dans le système de la figure 10 : pour tout $i > 0$, C_i est une prémisse de (res′) dont C_{i-1} est la conclusion. À toute branche de réfutation, on peut associer une séquence de littéraux $\ell_0, \ell_1, \dots$ telle que $\mathcal{P}_0(\ell_{i-1})$ soit le résolvant de l'application de (res′) de conclusion C_{i-1} et de prémisse $C_i = C_{i-1}, \ell_{i-1}$. Une *branche d'échec* est une branche de réfutation finie $C_0, C_1, \dots, C_n$ pour laquelle (sub) s'applique à C_n ; une dérivation de $S \vdash_{\mathbf{Rf}} \bot$ est donc telle que toutes ses branches sont des branches d'échec.

💬 *Une branche d'échec non redondante correspond exactement à une branche d'un arbre sémantique fermé pour S. Bien qu'on ne le montre pas ici, on a un analogue du lemme 10.8 : les réfutations dans le système de la figure 10 sont elles aussi « isomorphes » aux arbres sémantiques pour S ; ce système a justement été construit sur cette intuition.*

Une branche de réfutation est *non redondante* si pour tout $0 \leq i < j$, $\mathcal{P}_0(\ell_i) \neq \mathcal{P}_0(\ell_j)$, c'est-à-dire si l'on n'a pas jamais résolu sur la même proposition ; par extension, une dérivation de $S \vdash_{\mathbf{Rf}} \bot$ est non redondante si toutes ses branches sont non redondantes.

On dira enfin qu'une dérivation de $S \vdash_{\mathbf{Rf}} \bot$ *progresse* si, pour toutes ses branches de réfutation $C_0, C_1, \dots$ et pour tout i, $C_i \neq C_{i+1}$. Clairement, s'il existe une dérivation de $S \vdash_{\mathbf{Rf}} \bot$, alors il en existe une qui progresse. On peut aussi voir que, pour toute application de (res′) de résolvant P dans une dérivation qui progresse, $P \notin C$ et $\neg P \notin C$, sans quoi au moins une des prémisses serait égale à la conclusion. Une dérivation qui progresse est donc non redondante, et on a un énoncé similaire au corollaire 10.7.

Corollaire A.4. *La recherche de réfutation pour un ensemble fini F de clauses est dans* coNP.

Démonstration. La recherche de réfutation peut être effectuée par une machine de Turing alternante où les états existentiels servent à choisir le résolvant et les états universels à choisir la prémisse sur laquelle poursuivre la recherche d'une branche d'échec. Par le corollaire A.2, les états existentiels peuvent être remplacés par des choix déterministes. De plus, on peut se restreindre à la recherche de branches d'échec non redondantes, qui sont de longueur au plus $|\mathcal{P}_0(F)|$. C'est donc une machine dans coNP. □

Références

Sanjeev Arora et Boaz Barak, 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, UK. doi:10.1017/CBO9780511804090.

Yuri Breitbart, Harry B. Hunt III et Daniel J. Rosenkrantz, 1995. On the size of binary decision diagrams representing Boolean functions. *Theoretical Computer Science*, 145(1–2):45–69. doi:10.1016/0304-3975(94)00181-H.

Olivier Carton, 2008. *Langages formels, calculabilité et complexité*. Vuibert, Paris.

Chin-Liang Chang et Richard Char-Tung Lee, 1973. *Symbolic Logic and Mechanical Theorem Proving*. Computer Science and Applied Mathematics. Academic Press, New York. doi:10.1016/B978-0-08-091728-3.50001-1.

René David, Karim Nour et Christophe Raffalli, 2003. *Introduction à la logique*. Dunod, Paris, 2e édition.

Jean Goubault-Larrecq et Ian Mackie, 1997. *Proof Theory and Automated Deduction*, volume 6 de *Applied Logic Series*. Kluwer Academic Publishers, Amsterdam.

Michael A. Harrison, 1978. *Introduction to Formal Language Theory*. Addison-Wesley.

Donald E. Knuth, 2008. *The Art of Computer Programming*, volume 4, fascicle 0: *Introduction to Combinatorial Algorithms and Boolean Functions*. Addison-Wesley. URL https://cs.stanford.edu/~knuth/fasc0b.ps.gz.

Donald E. Knuth, 2011. *The Art of Computer Programming*, volume 4, fascicle 1: *Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley. URL https://cs.stanford.edu/~knuth/fasc1b.ps.gz.

Donald E. Knuth, 2015. *The Art of Computer Programming*, volume 4, fascicle 6: *Satisfiability*. Addison-Wesley. URL https://cs.stanford.edu/~knuth/fasc6a.ps.gz.

Richard Lassaigne et Michel de Rougemont, 2004. *Logic and Complexity*. Discrete Mathematics and Theoretical Computer Science. Springer, London. doi:10.1007/978-0-85729-392-3.

Harry R. Lewis, 1979. Satisfiability problems for propositional calculi. *Mathematical systems theory*, 13(1):45–53. doi:10.1007/BF01744287.

Christos Papadimitriou, 1993. *Computational Complexity*. Addison-Wesley.

Sylvain Perifel, 2014. *Complexité algorithmique*. Ellipses, Paris. URL https://www.irif.fr/~sperifel/complexite.pdf.

Seppo Sippu et Eljas Soisalon-Soininen, 1988. *Parsing Theory, Volume I: Languages and Parsing*. Springer-Verlag, Berlin Heidelberg.

Ingo Wegener, 2000. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM Monographs in Discrete Mathematics and Applications. SIAM, Philadelphia, USA. doi:10.1137/1.9780898719789.

Glynn Winskel, 1993. *The Formal Semantics of Programming Languages*. Foundations of Computing. MIT Press, Cambridge, Massachusetts, USA.