CHAPITRE VI

Logique propositionnelle

RÉSUMÉ. • *Une logique formelle consiste en un ensemble de* formules, *généralement des mots, avec une notion syntaxique de* preuves, *généralement des suites de formules obéissant à des* règles de déduction, *et une notion sémantique de* valeur, *déterminée à l'aide de* réalisations *convenables.*
• *Les formules de la logique propositionnelle* $\mathcal{L}_\bullet$ *sont des assemblages de variables* $\boldsymbol{X}_i$ *à l'aide de connecteurs* $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.
• *L'*évaluation *d'une formule de* $\mathcal{L}_\bullet$ *se fait inductivement à partir de l'*affectation *de valeurs* 0/1 *aux variables. Une formule est dite* valide *(resp.* satisfaisable*) si sa valeur est* 1 *pour toute affectation (resp. pour au moins une affectation).*
• *On dit que* H *se déduit* par coupure *à partir de* F *et* G *si* G *est la formule* F⇒H. *Une* preuve par coupure *est une suite de formules dont chacune est soit un* axiome *pris dans un liste fixée de 14 types de formules, soit obtenue par coupure à partir de formules antérieures de la liste.*
• *Le théorème de complétude affirme qu'une formule de* $\mathcal{L}_\bullet$ *est valide si et seulement si elle est prouvable.*
• *Le théorème de compacité affirme qu'un ensemble* T *de formules de* $\mathcal{L}_\bullet$ *est satisfaisable (c'est-à-dire qu'il existe une affectation de valeurs* 0/1 *qui rende vraie chaque formule de* T*) si et seulement si tout sous-ensemble fini de* T *l'est.*
• *De nombreux problèmes peuvent être codés en un problème de satisfaisabilité (ou de validité) pour un ensemble de formules propositionnelles. Le théorème de Cook et Levin affirme le caractère NP-complet de l'ensemble* SAT *des formules satisfaisables.*

▶ Ce chapitre est consacré à la logique propositionnelle booléenne, qui correspond à la pratique de la déduction sur des énoncés ne pouvant prendre que les valeurs « vrai » ou « faux ».

La première section décrit brièvement le cadre général des logiques formelles. Ensuite, on introduit la logique propositionnelle booléenne, pour laquelle on décrit à la fois une notion syntaxique de prouvabilité, basée sur l'utilisation de règles de déduction naturelles, et une notion sémantique de validité, basée sur l'affectation de valeurs booléennes aux variables. Le résultat principal est le théorème de complétude garantissant qu'une formule propositionnelle est valide si et seulement si elle est prouvable. Enfin, on étudie dans la dernière section le pouvoir d'expression de la logique booléenne et on établit en particulier le théorème de Cook et Levin qui exprime que le problème de reconnaître si un ensemble de formules propositionnelles est satisfaisable est NP-complet. Deux extensions de la logique booléenne, la logique intuitionniste et la logique linéaire, sont évoquées brièvement en appendice. ◀

▷ *La logique booléenne est la plus simple des logiques, et son étude est donc un moyen commode de se familiariser avec les notions de base et les principes importants sans avoir à affronter de grandes difficultés techniques. Un des points-clés est la distinction entre les notions syntaxiques,*

ne mettant en jeu que les formules de la logique, donc des mots finis, et les notions sémantiques, qui mettent en jeu une évaluation faisant référence à un contexte externe au monde des formules.

Techniquement assez facile, le théorème de complétude est néanmoins important, car il affirme que les quelques règles de déduction intervenant dans la définition des preuves épuisent les possibilités du raisonnement booléen. Il est par ailleurs l'archétype de résultats plus sophistiqués, en particulier le théorème de complétude pour les logiques du premier ordre qui sera établi au chapitre suivant.

La logique propositionnelle booléenne peut apparaître rudimentaire. L'intérêt des résultats de la section 4 et notamment du théorème de Cook et Levin est de montrer qu'il n'en est rien et que la famille des problèmes pouvant se coder dans cette logique est en fait beaucoup plus vaste que ce qu'on pourrait d'abord attendre. ◁

1. Logiques formelles: syntaxe et sémantique

► On discute brièvement un cadre général pour les logiques formelles, dans lequel entrent les diverses logiques qu'on considérera ensuite. ◄

▷ *Il n'existe pas* ***une*** *logique mathématique unique, mais* ***des*** *logiques, correspondant à des choix variés tant quant à la construction des formules qu'aux systèmes de preuve et à la notion de vérité. On propose dans cette section préliminaire un cadre général de logique formelle.*

D'une façon générale, une logique formelle $\mathcal{L}$ *est constituée d'une famille de formules, d'une notion de preuve et/ou d'une notion de vérité : les* ***formules*** *de* $\mathcal{L}$ *sont les objets sur lesquels portent les raisonnements, et sont habituellement des mots formés sur un certain alphabet : les* ***preuves*** *correspondent aux raisonnements valides, et sont généralement des suites de formules — voire des systèmes plus complexes : graphes, réseaux, etc. — obéissant à des règles syntaxiques particulières ; la* ***validité*** *d'une formule est généralement définie en faisant référence à une interprétation à l'aide de structures extérieures au monde des formules. Traditionnellement, on appelle* ***syntaxe*** *d'une logique* $\mathcal{L}$ *tout ce qui concerne les formules et les preuves, et* ***sémantique*** *tout ce qui fait appel à des interprétations externes — ou éventuellement internes : dans certains cas, les interprétations ont lieu à l'intérieur-même du monde des formules — donc en particulier ce qui concerne la notion de validité.*

Il apparaît donc une dualité entre une notion interne de ***prouvabilité*** $\vdash_{\mathcal{L}} \mathsf{F}$ *(« la formule* F *est prouvable dans la logique* $\mathcal{L}$ *») définie en termes syntaxiques, c'est-à-dire par des manipulations sur les mots, des régles de récriture, etc. , et une notion externe de* ***validité*** $\models_{\mathcal{L}} \mathsf{F}$ *(« la formule* F *est valide dans la logique* $\mathcal{L}$ *») définie en termes sémantiques, c'est-à-dire en attribuant un sens aux formules par référence à un monde souvent extérieur aux formules. De cette dualité résultent alors pour chaque logique* $\mathcal{L}$ *deux problèmes de compatibilité, à savoir le problème dit de* ***cohérence*** *:*

« *toute formule prouvable est-elle valide ?* »,

c'est-à-dire « $\vdash_{\mathcal{L}} \mathsf{F}$ *entraîne-t-elle* $\models_{\mathcal{L}} \mathsf{F}$ *?», et le problème dit de* ***complétude*** *:*

« *toute formule valide est-elle prouvable ?* »,

c'est-à-dire « $\models_{\mathcal{L}} \mathsf{F}$ *entraîne-t-elle* $\vdash_{\mathcal{L}} \mathsf{F}$ *?».*

A ce degré de généralité et donc d'imprécision il est difficile d'en dire beaucoup plus. Il existe de nombreuses logiques, certaines sont lacunaires au sens où elles possèdent une syntaxe mais pas de sémantique satisfaisante, c'est-à-dire garantissant cohérence et complétude — c'est en partie le cas de la logique intuitionniste dont les sémantiques sont satisfaisantes mais assez compliquées — d'autres au contraire admettent une sémantique naturelle mais pas de syntaxe correspondante : ainsi qu'on le verra au chapitre suivant, c'est notamment le cas des logiques du second ordre.

Dans les logiques considérées dans ce texte, les preuves sont toutes bâties sur le même schéma [1]*, à savoir comme suites de formules enchaînées à l'aide de règles de déduction se présentant comme des règles de récriture.* ◁

[1] mais il existe des logiques où les preuves sont des objets très différents, par exemple la Ω-logique de Woodin en théorie des ensembles où les preuves sont des ensembles (infinis) de réels

DÉFINITION 1.1. (preuve) Soit $\mathcal{L}$ une logique formelle, $\mathcal{R}$ une famille de règles de déduction, consistant en des applications partielles (effectives) associant à une suite finie de formules de $\mathcal{L}$ une formule de $\mathcal{L}$, T un ensemble de formules de $\mathcal{L}$, et F une formule de $\mathcal{L}$. On appelle *preuve* par $\mathcal{R}$ de F à partir de T dans $\mathcal{L}$ une suite finie $\mathsf{F}_1, \mathsf{F}_2, ..., \mathsf{F}_n$ de formules de $\mathcal{L}$ telle que F_n est F et, pour tout i, ou bien F_i est dans T, ou bien F_i s'obtient au moyen d'une des règles de $\mathcal{R}$ à partir de formules $\mathsf{F}_{j_1}, ..., \mathsf{F}_{j_p}$ avec $j_1, ..., j_p < i$. On dit que F est *prouvable* par $\mathcal{R}$ à partir de T, et on note $\mathsf{T} \vdash_{\mathcal{L},\mathcal{R}} \mathsf{F}$, ou simplement $\mathsf{T} \vdash \mathsf{F}$, s'il existe au moins une preuve par $\mathcal{R}$ de F à partir de T.

▷ *L'idée est que* T *constitue l'ensemble des hypothèses,* F *étant la formule à démontrer, et* $\mathcal{R}$ *l'ensemble des règles de démonstration admises: dire qu'on a* $\mathsf{T} \vdash_{\mathcal{L},\mathcal{R}} \mathsf{F}$ *correspond bien alors à l'intuition suivant laquelle démontrer* F *à partir de* T *au moyen des règles* $\mathcal{R}$ *consiste à construire une suite finie d'énoncés où chacun est soit l'une des hypothèses, soit est lui-même déduit à partir d'énoncés antérieurs au moyen d'une règle de raisonnement.*

Pour peu qu'il existe dans $\mathcal{R}$ *des règles à zéro argument, qu'on appelle des* ***axiomes****, on peut en particulier considérer les formules* F *qui sont prouvables à partir d'un ensemble vide d'hypothèses, ce qu'on notera* $\vdash_{\mathcal{L},\mathcal{R}} \mathsf{F}$*, ou* $\vdash \mathsf{F}$*.*

Il est usuel d'associer à chaque preuve un diagramme qui illustre l'ordre des déductions, l'élément de base étant

$$\frac{\mathsf{F}_1 \quad \dots \quad \mathsf{F}_n}{\mathsf{F}}\ \mathcal{R}$$

qui indique que la formule F *est déduite des formules* F_1*, ...,* F_n *par une des règles de* $\mathcal{R}$*. Un cas particulier est le diagramme*

$$\frac{}{\mathsf{F}}\ \mathcal{R}$$

qui indique que F *est un axiome de* $\mathcal{R}$*, autrement dit un point initial.*

La remarque suivante est importante. Le vocabulaire des logiques formelles calque celui de la logique utilisée dans les raisonnements mathématiques, ce qui est naturel puisque les unes sont des sortes de modèles mathématiques pour l'autre. Pour autant, cela ne doit pas ouvrir des abîmes de perplexité : démontrer des résultats de logique est exactement analogue à démontrer des résultats d'arithmétique ou de géométrie. Que les objets sur lesquels on raisonne s'appellent formules ou preuves au lieu de nombres entiers ou droites parallèles ne change rien à la nature des raisonnements effectués — laquelle reflète ce qu'on peut appeler le cadre métamathématique ambiant, ou simplement le niveau du discours. Lorsqu'on considère par exemple le problème de complétude « $\models_{\mathcal{L}} \mathsf{F}$ entraîne-t-elle $\vdash_{\mathcal{L}} \mathsf{F}$? » pour une certaine logique formelle $\mathcal{L}$*, le mot « entraînent » réfère à la logique usuelle, métamathématique, et non à la logique formelle* $\mathcal{L}$ *— pour laquelle ils ne veulent rien dire : les relations* $\models_{\mathcal{L}}$ *et* $\vdash_{\mathcal{L}}$ *mettent en jeu des formules de* $\mathcal{L}$ *et, éventuellement, d'utres objets, mais ne sont pas elles-mêmes des formules de* $\mathcal{L}$*, et, à ce titre, ne sauraient faire l'objet d'une preuve de* $\mathcal{L}$*.* ◁

2. La logique booléenne

► On définit la syntaxe et la sémantique de la logique propositionnelle booléenne. ◄

▷ *La suite de ce chapitre est consacrée à l'étude d'une logique formelle particulière notée* $\mathcal{L}_\bullet$ *et appelée logique propositionnelle booléenne, ou simplement logique booléenne, ou calcul booléen. Cette logique vise à modéliser les raisonnements usuels où les deux seules valeurs « vrai » et « faux » sont attribuées à des assertions, et où tout ce qui n'est pas vrai est faux et réciproquement. La logique ainsi introduite est ipso facto munie d'une sémantique issue de l'usage des connecteurs (conjonctions de coordination : et, ou, ...) qu'elle entend modéliser. La*

description de la syntaxe est ici limitée à celle des formules, l'introduction des preuves étant repoussée à la section suivante. ◁

2.1. Formules propositionnelles.

► On commence par décrire la syntaxe de la logique propositionnelle, c'est-à-dire la famille des formules. ◄

▷ *Les logiques propositionnelles sont des logiques dans lesquelles on prend seulement en compte l'assemblage de propositions, sans analyser plus avant le contenu de celles-ci. Pour cela, on introduit comme points de départ des variables représentant des propositions considérées comme atomiques, et on appelle formules les expressions obtenues en reliant des variables propositionnelles à l'aide de connecteurs simples correspondant aux opérations « négation », « conjonction », « disjonction », « implication », et « équivalence ».* ◁

DÉFINITION 2.1. (formule) On fixe une liste infinie de variables $\boldsymbol{X}_1, \boldsymbol{X}_2, \dots$ indexée par les entiers naturels (non nuls). On définit l'ensemble des *formules propositionnelles*, ou formules de $\mathcal{L}_\bullet$, comme le plus petit ensemble de mots contenant les variables $\boldsymbol{X}_\mathsf{i}$ et tel que, si F, G sont des formules, il en est de même de $\neg(\mathsf{F})$, lu « non F », de $(\mathsf{F})\wedge(\mathsf{G})$, lu « F et G », de $(\mathsf{F})\vee(\mathsf{G})$, lu « F ou G », de $(\mathsf{F})\Rightarrow(\mathsf{G})$, lu « F implique G » [2].

Ainsi, les mots $\boldsymbol{X}_2$, $(\boldsymbol{X}_1)\wedge(\neg(\boldsymbol{X}_2))$, $((\boldsymbol{X}_1)\vee(\boldsymbol{X}_3))\Rightarrow(\boldsymbol{X}_2)$ sont des formules propositionnelles.

▷ *Comme dans le cas des expressions arithmétiques, le rôle des parenthèses est d'éviter les ambiguïtés de lecture, et il est usuel de les supprimer autour des lettres isolées ou des négations, notant ainsi $\boldsymbol{X}_1\wedge\neg\boldsymbol{X}_2$ et $(\boldsymbol{X}_1\vee\boldsymbol{X}_3)\Rightarrow\boldsymbol{X}_2$ les formules ci-dessus* [3]*, ou encore $\mathsf{F}\wedge\mathsf{G}$ pour $(\mathsf{F})\wedge(\mathsf{G})$. On s'autorise aussi à utiliser d'autres lettres que $\boldsymbol{X}$ comme variable, ce qui revient à utiliser des métavariables, c'est-à-dire des variables représentant une autre variable.* ◁

La définition de l'ensemble des formules propositionnelles comme « plus petit ensemble tel que... » fournit immédiatement un principe d'induction et une notion de complexité sur les formules propositionnelles :

PROPOSITION 2.2. (induction) *Pour montrer qu'une propriété $\mathcal{P}$ est vraie pour toute formule propositionnelle, il suffit de montrer*

- *que $\mathcal{P}$ est vraie pour toute variable $\boldsymbol{X}_\mathsf{i}$,*
- *que, si $\mathcal{P}$ est vraie pour F, alors elle est vraie aussi pour $\neg\mathsf{F}$,*
- *et que, si $\mathcal{P}$ est vraie pour F et G, alors elle est vraie aussi pour $\mathsf{F}\wedge\mathsf{G}$, $\mathsf{F}\vee\mathsf{G}$, et $\mathsf{F}\Rightarrow\mathsf{G}$.*

[2] ce qui revient à dire que l'ensemble des formules de $\mathcal{L}_\bullet$ est le langage algébrique défini par la grammaire $S \to \boldsymbol{X}_1|\boldsymbol{X}_2|\dots|\neg(S)|(S)\wedge(S)|(S)\vee(S)|(S)\Rightarrow(S)$

[3] ce qui revient à remplacer la grammaire de la note précédente par $S \to A|\neg A|A\wedge A|A\vee A|A\Rightarrow A$, $A \to \boldsymbol{X}_1|\boldsymbol{X}_2|\dots|(S)$. Par ailleurs, d'un point de vue pratique, notamment pour une implémentation, il est déraisonnable de considérer les variables $\boldsymbol{X}_\mathsf{i}$ comme atomiques (faisant ainsi appel à un alphabet infini), et plus logique de les considérer elles-mêmes comme des mots écrits sur un alphabet, fini, composé de la lettre $\boldsymbol{X}$ et des dix lettres-indices $_0, \dots, _9$, de sorte que $\boldsymbol{X}_{13}$ est un mot de longueur 3, ce qui revient à invoquer la grammaire, cette fois finie, $S \to A|\neg A|A\wedge A|A\vee A|A\Rightarrow A$, $A \to V|(S)$, $V \to \boldsymbol{X}_1|\boldsymbol{X}_2|\dots|\boldsymbol{X}_9|V_0|V_1|\dots|V_9$. Sans importance ici, ce type de distinction est nécessaire lorsque des questions de complexité algorithmique entrent en jeu.

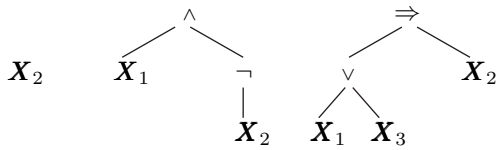
DÉMONSTRATION. Soit A l'ensemble des formules pour lesquelles $\mathcal{P}$ est vraie. Alors, par hypothèse, A contient les variables et est clos par négation, conjonction, disjonction, implication, et équivalence. Comme l'ensemble des formules est, par définition, le plus petit ensemble ayant les propriétés de clôture précédentes, A est l'ensemble de toutes les formules, autrement dit $\mathcal{P}$ est vraie pour toute formule. □

On déduit de ce qui précède que toute formule qui n'est pas une variable s'écrit de façon *unique* (aux parenthèses près) sous la forme $\neg\mathsf{F}$, ou $\mathsf{F}\wedge\mathsf{G}$, ou $\mathsf{F}\vee\mathsf{G}$, ou $\mathsf{F}\Rightarrow\mathsf{G}$, ou $\mathsf{F}\Leftrightarrow\mathsf{G}$. Il en résulte que, pour définir une application f sur les formules boolénnes, il suffit de définir $f(\boldsymbol{X}_\mathsf{i})$, puis de définir $f(\mathsf{F}\wedge\mathsf{G}), \ldots, f(\mathsf{F}\Leftrightarrow\mathsf{G})$ en fonction de $f(\mathsf{F})$ et $f(\mathsf{G})$. Ainsi, on construit l'ensemble des variables $\mathsf{Var}(\mathsf{F})$ et la *hauteur* $h(\mathsf{F})$ d'une formule F respectivement par

$$\mathsf{Var}(\boldsymbol{X}_\mathsf{i}) = \{\boldsymbol{X}_\mathsf{i}\}, \quad \mathsf{Var}(\neg\mathsf{F}) = \mathsf{Var}(\mathsf{F}), \quad \mathsf{Var}(\mathsf{F}\ \mathsf{c}\ \mathsf{G}) = \mathsf{Var}(\mathsf{F}) \cup \mathsf{Var}(\mathsf{G}),$$
$$h(\boldsymbol{X}_\mathsf{i}) = 0, \quad h(\neg\mathsf{F}) = h(\mathsf{F}) + 1, \quad h(\mathsf{F}\ \mathsf{c}\ \mathsf{G}) = \sup((h(\mathsf{F}), h(\mathsf{G})) + 1$$

pour $\mathsf{c} = \wedge, \vee, \Rightarrow$.

▷ *Une formule propositionnelle a une structure d'arbre : on associe un arbre étiqueté $a(\mathsf{F})$ à toute formule booléeenne en définissant $a(\boldsymbol{X}_\mathsf{i})$ comme constitué d'un unique sommet étiqueté $\boldsymbol{X}_\mathsf{i}$, $a(\neg\mathsf{F})$ comme l'arbre composé d'une racine étiquetée $\neg$ avec un unique descendant qui est l'arbre $a(\mathsf{F})$, et $a(\mathsf{F}\ \mathsf{c}\ \mathsf{G})$, pour $\mathsf{c} = \wedge, \vee, \Rightarrow$, comme l'arbre composé d'une racine étiquetée c avec deux descendants, à savoir à gauche l'arbre $a(\mathsf{F})$ et à droite l'arbre $a(\mathsf{G})$. Par exemple, les arbres associés aux formules citées plus haut sont*



La notion de hauteur et l'unicité de la décomposition sont évidentes dans cette représentation par arbre. On notera que l'induction sur les formules de la proposition 2.2 est simplement une induction sur la hauteur des formules.

*Pour usuelle et compréhensible qu'elle soit, la définition 2.1 n'est pas complète, dans la mesure où on n'a **défini** ni les variables $\boldsymbol{X}_\mathsf{i}$, ni les divers symboles $\neg, \Rightarrow, \ldots$ Afin de lever toute ambiguïté, et aussi pour préparer le terrain pour les développements ultérieurs, il sera utile de définir les formules propositionnelles, et, plus généralement, tous les objets de la logique $\mathcal{L}_\bullet$, comme des ensembles, ou, plus exactement et sur le modèle de ce qu'on a fait pour les entiers et les autres objets mathématiques, d'associer à chaque tel objet de $\mathcal{L}_\bullet$ une contrepartie dans le monde des ensembles. Il s'agit donc en particulier d'associer à toute formule F de $\mathcal{L}_\bullet$ telle qu'introduite dans la définition 2.1 d'associer un ensemble $\underline{\mathsf{F}}$ qui la spécifie de façon non-ambiguë. Il est usuel de définir les mots sur un alphabet A comme des suites finies d'éléments de A, et donc, pour obtenir une représentation ensembliste pour toutes les formules, il suffit de fixer une telle représentation pour les variables et les symboles logiques. Le choix est contingent, et essentiellement sans importance. Il sera commode pour la suite d'utiliser des entiers pour représenter les symboles de base, et, par ailleurs, de rendre explicite la structure d'arbre des formules telle que décrite ci-dessus, plutôt que de la lisser sous la forme de mots comme l'introduction des parenthèses le permet.* ◁

DÉFINITION 2.3. (formule, *bis*) Pour chaque entier i non nul, on note $\underline{\boldsymbol{X}_i}$ pour $(0, i)$, et on pose $\underline{\neg} := 1$, $\underline{\Rightarrow} := 2$, $\underline{\vee} := 3$, $\underline{\wedge} := 4$. On définit l'ensemble des formules de $\underline{\mathcal{L}_\bullet}$, comme le plus petit ensemble de suites contenant $\underline{\boldsymbol{X}_i}$ pour tout i et tel que, si F, G sont des formules, il en est de même de $(\underline{\neg}, F)$, de $(\underline{\Rightarrow}, F, G)$, de $(\underline{\vee}, F, G)$, et de $(\underline{\wedge}, F, G)$.

▷ *À toute formule* F *de* $\mathcal{L}_\bullet$ *correspond une formule* $\underline{\mathsf{F}}$ *de* $\underline{\mathcal{L}_\bullet}$ *qui en est l'exacte contrepartie, et qui, par construction, est un ensemble, et, plus précisément un élément de* V_ω *puisqu'obtenu par itération finie de formation de suites finies. Par exemple, la contrepartie de la formule* $\boldsymbol{X}_2$ *est l'ensemble* $(0,2)$*, et celle de la formule* $\boldsymbol{X}_1 \wedge \neg \boldsymbol{X}_2$ *est l'ensemble* $(\wedge, \underline{\boldsymbol{X}_1}, (\neg, \underline{\boldsymbol{X}_2}))$*, c'est-à-dire* $(4,(0,1),(1,(0,2)))$*. Comme on l'a fait avec les entiers et les autres objets mathématiques, on omettra dans la suite la distinction entre une formule* F *et sa contrepartie ensembliste* $\underline{\mathsf{F}}$*.*

Il est usuel d'introduire la convention de notation suivante. ◁

NOTATION 2.4. (équivalence) Pour F, G formules propositionnelles, on note $\mathsf{F} \Leftrightarrow \mathsf{G}$, lu « F équivalent à G », la formule $(\mathsf{F}{\Rightarrow}\mathsf{G}){\wedge}(\mathsf{G}{\Rightarrow}\mathsf{F})$.

▷ *Rien n'empêche d'introduire le symbole* $\Leftrightarrow$ *comme un connecteur primitif supplémentaire et de faire de* $\mathsf{F} \Leftrightarrow \mathsf{G}$ *une formule à part entière et non une simple notation pour une autre formule. L'option retenue ici offre l'avantage de raccourcir les démonstrations ultérieures en limitant le nombre de connecteurs à considérer, et elle ne restreint en rien le pouvoir d'expression dès lors que la sémantique qu'on souhaite attribuer à* $\mathsf{F} \Leftrightarrow \mathsf{G}$ *coïncide avec celle de* $(\mathsf{F}{\Rightarrow}\mathsf{G}){\wedge}(\mathsf{G}{\Rightarrow}\mathsf{F})$*.*

On pourrait traiter $\wedge$ *et* $\vee$ *de la même manière et ne garder que* $\neg$ *et* $\Rightarrow$*, voire même seulement* $\Rightarrow$ *et une formule additionnelle* $\bot$ *représentant le faux.* ◁

2.2. Sémantique de la logique booléenne.

► On introduit la classique sémantique à deux valeurs de la logique booléenne. ◄

▷ *Définir une sémantique pour une logique, c'est attacher à chaque formule une interprétation faisant référence à un contexte externe. Suivant la logique considérée, cette interprétation peut prendre ses valeurs dans des espaces très divers. Dans le cas de la logique booléenne, l'interprétation consiste en une valeur de vérité qui est soit « vrai » soit « faux ». On la définit récursivement à partir d'une affectation de valeurs aux variables propositionnelles en calquant le sens usuel des connecteurs logiques.*

Le but du calcul booléen est de modéliser le raisonnement naturel — ou tout au moins l'un des aspects de celui-ci. Comme on considère ici le cas le plus simple, où seules deux valeurs de vérité sont introduites, « vrai » et « faux », alias 1 *et* 0*, la définition récursive d'une valeur de vérité pour chaque formule est dictée par l'usage des conjonctions « et », « ou », … du raisonnement naturel que les connecteurs logiques se proposent de mimer.*

Noter que l'évaluation d'une formule n'est pas absolue, mais dépend d'un contexte qui, en l'occurrence, consiste en l'affectation initiale de valeurs aux variables propositionnelles. Un tel contexte est aussi appelé ***réalisation*** *puisque c'est ce qui permet de passer du monde abstrait des formules au monde concret du « vrai–faux ».* ◁

DÉFINITION 2.5. (affectation, évaluation) On appelle affectation de valeurs booléennes, ou simplement *affectation*, pour une formule propositionnelle F une fonction (partielle) V de $\{\boldsymbol{X}_\mathsf{i}\,;\, \mathsf{i} \geqslant 1\}$ dans $\{0,1\}$ dont le domaine inclut $\mathsf{Var}(\mathsf{F})$; l'*évaluation* de F en V, notée $\mathrm{eval}_V(\mathsf{F})$, est alors définie récursivement par les clauses

$$\mathrm{eval}_V(\boldsymbol{X}_\mathsf{i}) = V(\boldsymbol{X}_\mathsf{i}),\ \mathrm{eval}_V(\neg\mathsf{F}) = f_\neg(\mathrm{eval}_V(\mathsf{F})),\ \mathrm{eval}_V(\mathsf{F}\mathsf{c}\mathsf{G}) = f_\mathsf{c}(\mathrm{eval}_V(\mathsf{F}), \mathrm{eval}_V(\mathsf{G})),$$

où les fonctions $f_\neg, f_\wedge, f_\vee, f_\Rightarrow$ sont déterminées par

$f_\neg$	
0	1
1	0

$f_\wedge$	0	1
0	0	0
1	0	1

$f_\vee$	0	1
0	0	1
1	1	1

$f_\Rightarrow$	0	1
0	1	1
1	0	1

L'existence et l'unicité de l'évaluation résultent de la proposition 2.2. Par exemple, l'évaluation de la formule $(\boldsymbol{X}_1 \vee \boldsymbol{X}_3) \Rightarrow \boldsymbol{X}_2$ en l'affectation V définie par $V(\boldsymbol{X}_1) = V(\boldsymbol{X}_3) = 1$, $V(\boldsymbol{X}_2) = 0$, aussi appelée évaluation de F en $\boldsymbol{X}_1 = \boldsymbol{X}_3 = 1$, $\boldsymbol{X}_2 = 0$, est

$$\mathrm{eval}_V((\boldsymbol{X}_1 \vee \boldsymbol{X}_3) \Rightarrow \boldsymbol{X}_2) = f_\Rightarrow(f_\vee(1,1),0) = f_\Rightarrow(1,0) = 0.$$

▷ *Sur la représentation par arbre, l'évaluation d'une formule consiste à remplacer les variables par leur valeur, puis à remonter de proche en proche les valeurs vers la racine au moyen des fonctions f_{c}. En particulier, au plus h étapes sont nécessaires pour une formule de hauteur h.*

Le choix des fonctions f_{c} associées aux différents connecteurs c n'est pas arbitraire, et il est dicté par l'usage projeté de la logique propositionnelle comme modèle du raisonnement naturel. On peut défendre le choix de $f_\Rightarrow$ comme étant le moins mauvais des choix dès lors que seules les deux valeurs 0 et 1 sont possibles. On peut aussi protester que ce choix, qui mène à déclarer vraies toutes les implications dont la prémisse est fausse, n'est pas une bonne modélisation de notre intuition de ce qu'est une implication. Ceci ne change rien au développement de la logique booléenne, mais peut suggérer d'introduire ultérieurement d'autres logiques rendant mieux compte de l'existence d'un lien entre prémisse et conclusion dans une implication. C'est typiquement ce que fait la logique intuitionniste, dont les formules sont les mêmes que celles de la logique propositionnelle, mais dont la sémantique et le système de preuve sont différents. ◁

Une fois les valeurs de vérité attribuées, les définitions suivantes s'accordent avec le sens commun :

Définition 2.6. (satisfaction, validité, équivalence) Soient F, G des formules propositionnelles, et T une famille de formules propositionnelles.
(*i*) On dit qu'une affectation V *satisfait* F, ou encore que F est *vraie* dans V, (*resp.* T), noté $V \models \mathsf{F}$ (*resp.* $V \models \mathsf{T}$), si on a $\mathrm{eval}_V(\mathsf{F}) = 1$ (*resp.* $\mathrm{eval}_V(\mathsf{F}) = 1$ pour chaque F dans T).
(*ii*) On dit que F est *valide*, noté $\models \mathsf{F}$, si toute affectation satisfait F, et de même pour T. On dit que F est *satisfaisable* s'il existe au moins une affectation satisfaisant F, et de même pour T.
(*iii*) On dit que F et G sont *équivalentes* si les affectations satisfaisant F et G sont les mêmes.

Par exemple, la formule $\boldsymbol{X}_1 \vee \neg \boldsymbol{X}_1$ est valide, puisque satisfaite par toute affectation V : en effet, il n'y a que deux possibilités pour $V(\boldsymbol{X}_1)$, à savoir $V(\boldsymbol{X}_1) = 0$ et $V(\boldsymbol{X}_1) = 1$, et les deux entraînent $V(\boldsymbol{X}_1 \vee \neg \boldsymbol{X}_1) = 1$, par définition de la fonction $f_\vee$.

Les résultats suivants se déduisent immédiatement des définitions :

Lemme 2.7. *Une formule $\neg\mathsf{F}$ est valide si et seulement si F n'est pas satisfaisable. Une formule $\mathsf{F} \wedge \mathsf{G}$ est valide si et seulement si F et G sont valides. Une formule $\mathsf{F} \Rightarrow \mathsf{G}$ est valide si et seulement si toute affectation satisfaisant F satisfait aussi G. Une formule $\mathsf{F} \Leftrightarrow \mathsf{G}$ est valide si et seulement si les formules F et G sont équivalentes.*

▷ *Même si la notion de décidabilité n'a pas encore été définie précisément — elle le sera dans la section 4, puis au chapitre VIII — le sens du résultat suivant doit être clair.* ◁

PROPOSITION 2.8. (décidabilité) *Il existe un algorithme décidant la satisfaisabilité (resp. la validité) des formules propositionnelles, c'est-à-dire qui, partant d'une formule quelconque* F*, décide en un nombre fini d'étapes si* F *est satisfaisable (resp. valide) ou non.*

DÉMONSTRATION. Pour chaque formule F, le nombre des variables ayant une occurrence dans F est fini. On peut donc construire la *table de vérité* de F en énumérant tous les choix possibles pour les valeurs des variables apparaissant dans F, et, pour chacun de ces choix V, évaluant F en V. On décide alors la satisfaisabilité (*resp.* la validité) de F en lisant dans la table de vérité si la valeur 1 est obtenue au moins une fois (*resp.* à chaque fois). □

3. Le théorème de complétude pour la logique booléenne

► On introduit la notion de preuve par coupure en logique booléenne, et on établit le théorème de complétude affirmant que toute formule valide est prouvable à partir d'une famille finie explicite d'axiomes simples. ◄

▷ *Une fois introduite une sémantique avec une notion de formule valide, se pose la question de reconnaître les formules valides. Dans le raisonnement naturel, la méthode usuelle pour reconnaître qu'une assertion est valide est de la* ***démontrer****, et on est ainsi mené à développer une ou des notions de preuves en logique boolénne. La question a deux faces : soit on privilégie la proximité avec le raisonnement naturel, et on est mené à des preuves telles que celles étudiées ci-dessous ; soit on privilégie l'efficacité algorithmique, et des preuves d'un autre type, en particulier les preuves par résolution, apparaissent. Dans tous les cas, se pose le problème de la complétude du système de preuve concerné, qui est la question de savoir si toutes les formules valides sont prouvables. Ici est traité le cas des preuves par coupure — le cas des preuves par résolution est abordé dans l'exercice 6.* ◁

3.1. Preuves par coupure.

► On introduit la notion de preuve par coupure (ou *modus ponens*). ◄

▷ *Dans le raisonnement naturel, il existe plusieurs schémas de déduction, ou syllogismes. Dans la suite de cette partie, on considère une unique règle de déduction, c'est-à-dire si on veut une unique méthode de preuve.* ◁

DÉFINITION 3.1. (coupure) Soient $\mathsf{F}, \mathsf{G}, \mathsf{H}$ des formules propositionnelles. On dit que H est déduite *par coupure* à partir de F et G si G est la formule $\mathsf{F}{\Rightarrow}\mathsf{H}$.

Par exemple, supposons $\mathsf{F} = \boldsymbol{X}_1{\vee}\boldsymbol{X}_2$ et $\mathsf{G} = (\boldsymbol{X}_1{\vee}\boldsymbol{X}_2){\Rightarrow}\boldsymbol{X}_1$. Alors $\boldsymbol{X}_1$ se déduit par coupure à partir de F et G. Suivant le schéma général indiqué dans la section 1, la règle de coupure correspond au diagramme

$$\frac{\mathsf{F} \qquad \mathsf{F}{\Rightarrow}\mathsf{H}}{\mathsf{H}}\ \text{coupure}$$

indiquant qu'on déduit la conclusion H par coupure à partir de F et de $\mathsf{F}{\Rightarrow}\mathsf{H}$.

L'utilisation de la coupure seule ne permet pas de construire de preuve *ex nihilo*. On lui adjoint des règles sans argument (c'est-à-dire des fonctions à zéro variable sur l'ensemble des formules), appelées *axiomes*, permettant de poser une formule comme point de départ.

DÉFINITION 3.2. (instance, axiomes) (*i*) Soient F, G des formules propositionnelles. On dit que G est une *instance* de F s'il existe des formules G_i telle que G est obtenue en remplaçant $\boldsymbol{X}_\mathsf{i}$ par G_i pour chaque i.

(*ii*) On appelle *axiome de la logique booléenne* toute instance d'une des formules suivantes:
(axiomes pour l'implication)
A_1: $\boldsymbol{X}_1 \Rightarrow (\boldsymbol{X}_2 \Rightarrow \boldsymbol{X}_1)$, A_2: $(\boldsymbol{X}_1 \Rightarrow (\boldsymbol{X}_2 \Rightarrow \boldsymbol{X}_3)) \Rightarrow ((\boldsymbol{X}_1 \Rightarrow \boldsymbol{X}_2) \Rightarrow (\boldsymbol{X}_1 \Rightarrow \boldsymbol{X}_3))$
(axiomes pour la négation)
A_3: $\boldsymbol{X}_1 \Rightarrow \neg\neg \boldsymbol{X}_1$, A_4: $\neg\neg \boldsymbol{X}_1 \Rightarrow \boldsymbol{X}_1$, A_5: $(\boldsymbol{X}_1 \Rightarrow \boldsymbol{X}_2) \Rightarrow (\neg \boldsymbol{X}_2 \Rightarrow \neg \boldsymbol{X}_1)$
(axiomes pour la conjonction)
A_6: $\boldsymbol{X}_1 \Rightarrow (\boldsymbol{X}_2 \Rightarrow (\boldsymbol{X}_1 \wedge \boldsymbol{X}_2))$, A_7: $(\boldsymbol{X}_1 \wedge \boldsymbol{X}_2) \Rightarrow \boldsymbol{X}_1$, A_8: $(\boldsymbol{X}_1 \wedge \boldsymbol{X}_2) \Rightarrow \boldsymbol{X}_2$
(axiomes pour la disjonction)
A_9: $\boldsymbol{X}_1 \Rightarrow (\boldsymbol{X}_1 \vee \boldsymbol{X}_2)$, A_{10}: $\boldsymbol{X}_2 \Rightarrow (\boldsymbol{X}_1 \vee \boldsymbol{X}_2)$, A_{11}: $\neg \boldsymbol{X}_1 \Rightarrow ((\boldsymbol{X}_1 \vee \boldsymbol{X}_2) \Rightarrow \boldsymbol{X}_2)$

Suivant le schéma général de la section 1, on a une notion de preuve par coupure et axiomes :

DÉFINITION 3.3. (preuve) Soient T une famille de formules propositionnelles, et F une formule propositionnelle. On appelle *preuve par coupure (et axiomes)* de F à partir de T une suite finie $\mathsf{F}_1, \mathsf{F}_2, \ldots, \mathsf{F}_\mathsf{n}$ de formules propositionnelles telle que F_n est égale à F et, pour tout i, ou bien F_i est dans T, ou bien F_i est un axiome, ou bien F_i s'obtient par coupure à partir de deux formules $\mathsf{F}_\mathsf{j}, \mathsf{F}_\mathsf{k}$ avec $\mathsf{j}, \mathsf{k} < \mathsf{i}$. On note $\mathsf{T} \vdash \mathsf{F}$ si F est prouvable par coupure (et axiomes) à partir de T ; on note $\vdash \mathsf{F}$ pour $\emptyset \vdash \mathsf{F}$.

EXEMPLE 3.4. (preuve) Soient $\mathsf{F}, \mathsf{G}, \mathsf{H}$ trois formules propositionnelles. Alors la suite ci-dessous est une preuve par coupure de $\mathsf{F} \Rightarrow \mathsf{H}$ à partir de $\{\mathsf{F} \Rightarrow \mathsf{G}, \mathsf{G} \Rightarrow \mathsf{H}\}$:

$\mathsf{F}_1 : \mathsf{G} \Rightarrow \mathsf{H}$ (hypothèse)
$\mathsf{F}_2 : (\mathsf{G} \Rightarrow \mathsf{H}) \Rightarrow (\mathsf{F} \Rightarrow (\mathsf{G} \Rightarrow \mathsf{H}))$ (instance de A_1)
$\mathsf{F}_3 : \mathsf{F} \Rightarrow (\mathsf{G} \Rightarrow \mathsf{H})$ (coupure à partir de F_1 et F_2)
$\mathsf{F}_4 : (\mathsf{F} \Rightarrow (\mathsf{G} \Rightarrow \mathsf{H})) \Rightarrow ((\mathsf{F} \Rightarrow \mathsf{G}) \Rightarrow (\mathsf{F} \Rightarrow \mathsf{H}))$ (instance de A_2)
$\mathsf{F}_5 : (\mathsf{F} \Rightarrow \mathsf{G}) \Rightarrow (\mathsf{F} \Rightarrow \mathsf{H})$ (coupure à partir de F_3 et F_4)
$\mathsf{F}_6 : \mathsf{F} \Rightarrow \mathsf{G}$ (hypothèse)
$\mathsf{F}_7 : \mathsf{F} \Rightarrow \mathsf{H}$. (coupure à partir de F_6 et F_5)

On a donc $\{\mathsf{F} \Rightarrow \mathsf{G}, \mathsf{G} \Rightarrow \mathsf{H}\} \vdash \mathsf{F} \Rightarrow \mathsf{H}$ (voir figure 1).

3.2. Le théorème de la déduction.

▶ On relie prouvabilité et implication. ◀

▷ *La démonstration du fait que toute formule valide est prouvable par coupure qui sera donnée dans la section suivante repose sur plusieurs résultats préliminaires, dont le théorème dit de la déduction. Reposant sur des vérifications syntaxiques simples, la démonstration de celui-ci est facile, mais elle permetra surtout de commencer à manipuler des preuves.* ◁

$$\dfrac{\dfrac{\dfrac{\mathsf{F}\Rightarrow\mathsf{G}\qquad\dfrac{\mathsf{G}\Rightarrow\mathsf{H}\qquad\dfrac{}{(\mathsf{G}\Rightarrow\mathsf{H})\Rightarrow(\mathsf{F}\Rightarrow(\mathsf{G}\Rightarrow\mathsf{H}))}\,\text{axiome}}{\mathsf{F}\Rightarrow(\mathsf{G}\Rightarrow\mathsf{H})}\,\text{coupure}\qquad\dfrac{}{(\mathsf{F}\Rightarrow(\mathsf{G}\Rightarrow\mathsf{H}))\Rightarrow((\mathsf{F}\Rightarrow\mathsf{G})\Rightarrow(\mathsf{F}\Rightarrow\mathsf{H}))}\,\text{axiome}}{(\mathsf{F}\Rightarrow\mathsf{G})\Rightarrow(\mathsf{F}\Rightarrow\mathsf{H})}\,\text{coupure}}{\mathsf{F}\Rightarrow\mathsf{H}}\,\text{coupure}$$

FIGURE 1. Preuve de $\mathsf{F}\Rightarrow\mathsf{H}$ à partir de $\mathsf{F}\Rightarrow\mathsf{G}$ et $\mathsf{G}\Rightarrow\mathsf{H}$: on peut voir une preuve, ici celle de l'exemple 3.4, comme un arbre dont la racine est la formule à prouver et les feuilles sont les hypothèses et les axiomes.

PROPOSITION 3.5. (théorème de la déduction) *Soit* T *une famille de formules propositionnelles, et* F, G *deux formules propositionnelles. Alors* $\mathsf{T}\vdash\mathsf{F}\Rightarrow\mathsf{G}$ *équivaut à* $\mathsf{T}\cup\{\mathsf{F}\}\vdash\mathsf{G}$.

DÉMONSTRATION. Une direction est triviale: si $\mathsf{H}_1,\dots,\mathsf{H}_\mathsf{n}$ est une preuve de $\mathsf{F}\Rightarrow\mathsf{G}$ à partir de T, alors $\mathsf{H}_1,\dots,\mathsf{H}_\mathsf{n}$, F, G est une preuve de G à partir de $\mathsf{T}\cup\{\mathsf{F}\}$. Inversement, on montre par récurrence sur n que, s'il existe une preuve de longueur n de G à partir de $\mathsf{T}\cup\{\mathsf{F}\}$, alors il existe une preuve de $\mathsf{F}\Rightarrow\mathsf{G}$ à partir de T. Supposons que $\mathsf{H}_1,\dots,\mathsf{H}_\mathsf{n}$ est une preuve de G à partir de $\mathsf{T}\cup\{\mathsf{F}\}$. Par hypothèse de récurrence, il existe une preuve à partir de T pour chacune des formules $\mathsf{F}\Rightarrow\mathsf{H}_1,\dots,\mathsf{F}\Rightarrow\mathsf{H}_{\mathsf{n}-1}$ — ceci s'appliquant par défaut au cas $\mathsf{n}=1$. Considérons alors la formule H_n, c'est-à-dire G. Trois cas sont possibles.
(*i*) La formule G est un axiome, ou une formule de T. On a alors $\mathsf{T}\vdash\mathsf{G}$; par ailleurs, $\mathsf{G}\Rightarrow(\mathsf{F}\Rightarrow\mathsf{G})$ est une instance de l'axiome A_1, donc on a $\vdash\mathsf{G}\Rightarrow(\mathsf{F}\Rightarrow\mathsf{G})$, et *a fortiori* $\mathsf{T}\vdash\mathsf{G}\Rightarrow(\mathsf{F}\Rightarrow\mathsf{G})$. Par coupure, on déduit $\mathsf{T}\vdash\mathsf{F}\Rightarrow\mathsf{G}$.
(*ii*) La formule G est la formule F. La suite

$(\mathsf{F}\Rightarrow((\mathsf{F}\Rightarrow\mathsf{F})\Rightarrow\mathsf{F}))\Rightarrow((\mathsf{F}\Rightarrow(\mathsf{F}\Rightarrow\mathsf{F}))\Rightarrow(\mathsf{F}\Rightarrow\mathsf{F}))$ (instance de A_2)
$\mathsf{F}\Rightarrow((\mathsf{F}\Rightarrow\mathsf{F})\Rightarrow\mathsf{F})$ (instance de A_1)
$(\mathsf{F}\Rightarrow(\mathsf{F}\Rightarrow\mathsf{F}))\Rightarrow(\mathsf{F}\Rightarrow\mathsf{F})$, (coupure)
$\mathsf{F}\Rightarrow(\mathsf{F}\Rightarrow\mathsf{F})$, (instance de A_1)
$\mathsf{F}\Rightarrow\mathsf{F}$ (coupure)

est une preuve. On a donc $\vdash\mathsf{F}\Rightarrow\mathsf{F}$, et *a fortiori*, $\mathsf{T}\vdash\mathsf{F}\Rightarrow\mathsf{F}$, soit $\mathsf{T}\vdash\mathsf{F}\Rightarrow\mathsf{G}$.
(*iii*) Il existe $\mathsf{i},\mathsf{j}<\mathsf{n}$ tels que H_j est une formule $\mathsf{H}_\mathsf{i}\Rightarrow\mathsf{G}$. Par hypothèse de récurrence, on a $\mathsf{T}\vdash\mathsf{F}\Rightarrow\mathsf{H}_\mathsf{i}$ et $\mathsf{T}\vdash\mathsf{F}\Rightarrow(\mathsf{H}_\mathsf{i}\Rightarrow\mathsf{G})$. Alors la suite

$(\mathsf{F}\Rightarrow(\mathsf{H}_\mathsf{i}\Rightarrow\mathsf{G}))\Rightarrow((\mathsf{F}\Rightarrow\mathsf{H}_\mathsf{i})\Rightarrow(\mathsf{F}\Rightarrow\mathsf{G}))$ (instance de A_2)
$(\mathsf{F}\Rightarrow\mathsf{H}_\mathsf{i})\Rightarrow(\mathsf{F}\Rightarrow\mathsf{G})$ (coupure)
$\mathsf{F}\Rightarrow\mathsf{G}$ (coupure)

est une preuve de $\mathsf{F}\Rightarrow\mathsf{G}$ à partir de $\mathsf{F}\Rightarrow\mathsf{H}_\mathsf{i}$ et $\mathsf{F}\Rightarrow(\mathsf{H}_\mathsf{i}\Rightarrow\mathsf{G})$. En concaténant cette preuve à une preuve de $\mathsf{F}\Rightarrow\mathsf{H}_\mathsf{i}$ et de $\mathsf{F}\Rightarrow(\mathsf{H}_\mathsf{i}\Rightarrow\mathsf{G})$ à partir de T, on obtient une preuve de $\mathsf{F}\Rightarrow\mathsf{G}$ à partir de T. □

3.3. Le théorème de complétude.

► On montre qu'une formule propositionnelle est valide si et seulement si elle est prouvable par coupure. ◄

▷ *A ce point, on dispose de deux notions de vérité : une notion syntaxique fondée sur l'existence d'une preuve, et une notion sémantique fondée sur l'affectation de valeurs. On va montrer ici que ces deux notions coïncident. Même si la démonstration est techniquement aisée, il s'agit d'un résultat important et profond car il relie deux approches a priori disjointes.*

On va montrer successivement deux inclusions : la première, facile, affirme que tout ce qui est prouvable est valide (résultat dit de cohérence) ; la seconde, plus délicate, affirme que tout ce qui est valide est prouvable (résultat dit de complétude). ◁

PROPOSITION 3.6. (cohérence) *Toute formule propositionnelle prouvable est valide.*

DÉMONSTRATION. Une vérification immédiate montre que les formules $\mathsf{A}_1, \ldots, \mathsf{A}_{14}$ sont valides, et, d'autre part, que si F est valide, il en est de même de chacune des instances de F. Par ailleurs, la règle de coupure est valide, au sens où, si F et $\mathsf{F}{\Rightarrow}\mathsf{G}$ sont valides, il en est de même de G. Une induction sur n montre alors que, si F a une preuve de longueur au plus n, alors F est valide. □

PROPOSITION 3.7. (théorème de complétude, forme locale) *Toute formule propositionnelle valide est prouvable.*

La démonstration requiert du soin, et on commence par des résultats auxiliaires.

LEMME 3.8. (*i*) *Les relations* $\mathsf{T}\cup\{\mathsf{F}\}\vdash\mathsf{G}$ *et* $\mathsf{T}\cup\{\neg\mathsf{G}\}\vdash\neg\mathsf{F}$ *sont équivalentes.*
(*ii*) *Si on a à la fois* $\mathsf{T}\vdash\mathsf{F}$ *et* $\mathsf{T}\vdash\neg\mathsf{F}$, *alors on a* $\mathsf{T}\vdash\mathsf{G}$ *pour toute formule* G.
(*iii*) *Si on a à la fois* $\mathsf{T}\cup\{\mathsf{F}\}\vdash\mathsf{G}$ *et* $\mathsf{T}\cup\{\neg\mathsf{F}\}\vdash\mathsf{G}$, *alors on a* $\mathsf{T}\vdash\mathsf{G}$.

DÉMONSTRATION. (*i*) Supposons $\mathsf{T}\cup\{\mathsf{F}\}\vdash\mathsf{G}$. Par le théorème de la déduction, on a $\mathsf{T}\vdash\mathsf{F}{\Rightarrow}\mathsf{G}$. Or la formule $(\mathsf{F}{\Rightarrow}\mathsf{G}){\Rightarrow}(\neg\mathsf{G}{\Rightarrow}\neg\mathsf{F})$ est une instance de l'axiome A_5, donc, par coupure, on obtient $\mathsf{T}\vdash\neg\mathsf{G}{\Rightarrow}\neg\mathsf{F}$, d'où, par le théorème de la déduction à nouveau, $\mathsf{T}\cup\{\neg\mathsf{G}\}\vdash\neg\mathsf{F}$.

Supposons maintenant $\mathsf{T}\cup\{\neg\mathsf{G}\}\vdash\neg\mathsf{F}$. Appliquant ce qui précède, on obtient $\mathsf{T}\cup\{\neg\neg\mathsf{F}\}\vdash\neg\neg\mathsf{G}$. Ensuite, $\neg\neg\mathsf{G}{\Rightarrow}\mathsf{G}$ est une instance de l'axiome A_3, donc, par coupure, on déduit $\mathsf{T}\cup\{\neg\neg\mathsf{F}\}\vdash\mathsf{G}$. Enfin, $\mathsf{F}{\Rightarrow}\neg\neg\mathsf{F}$ est une instance de l'axiome A_4: de là, on déduit de toute preuve d'une formule à partir de $\mathsf{T}\cup\{\neg\neg\mathsf{F}\}$ une preuve de la même formule à partir de $\mathsf{T}\cup\{\mathsf{F}\}$, et on obtient finalement $\mathsf{T}\cup\{\mathsf{F}\}\vdash\mathsf{G}$.

(*ii*) On a $\{\neg\mathsf{F},\neg\mathsf{G}\}\vdash\neg\mathsf{F}$ par définition, d'où, par (*i*), $\{\neg\mathsf{F},\mathsf{F}\}\vdash\mathsf{G}$, et, de là, $\mathsf{T}\vdash\mathsf{G}$ si à la fois F et $\neg\mathsf{F}$ sont prouvables à partir de T.

(*iii*) Supposons $\mathsf{T}\cup\{\mathsf{F}\}\vdash\mathsf{G}$ et $\mathsf{T}\cup\{\neg\mathsf{F}\}\vdash\mathsf{G}$. Appliquant (*i*), on obtient $\mathsf{T}\cup\{\neg\mathsf{G}\}\vdash\neg\mathsf{F}$ et $\mathsf{T}\cup\{\neg\mathsf{G}\}\vdash\neg\neg\mathsf{F}$. Par (*ii*), toute formule est donc prouvable à partir de $\mathsf{T}\cup\{\neg\mathsf{G}\}$, et on a en particulier $\mathsf{T}\cup\{\neg\mathsf{G}\}\vdash\mathsf{G}$, d'où, par déduction, $\mathsf{T}\vdash\neg\mathsf{G}{\Rightarrow}\mathsf{G}$. Il suffit alors d'établir la relation $\{\neg\mathsf{G}{\Rightarrow}\mathsf{G}\}\vdash\mathsf{G}$ pour déduire $\mathsf{T}\vdash\mathsf{G}$. Or, par définition, on a $\{\neg\mathsf{G},\neg\mathsf{G}{\Rightarrow}\mathsf{G}\}\vdash\mathsf{G}$, d'où, en appliquant (*i*), $\{\neg\mathsf{G},\neg\mathsf{G}\}\vdash\neg(\neg\mathsf{G}{\Rightarrow}\mathsf{G})$, soit $\{\neg\mathsf{G}\}\vdash\neg(\neg\mathsf{G}{\Rightarrow}\mathsf{G})$, d'où, en réappliquant (*i*) à l'envers, $\{\neg\mathsf{G}{\Rightarrow}\mathsf{G}\}\vdash\mathsf{G}$. □

LEMME 3.9. *Les relations suivantes sont toujours vérifiées:*
(*i*) $\{\mathsf{F}\}\vdash\neg\neg\mathsf{F}$;
(*ii*) $\{\mathsf{F},\mathsf{G}\}\vdash\mathsf{F}{\wedge}\mathsf{G}$, $\{\neg\mathsf{F}\}\vdash\neg(\mathsf{F}{\wedge}\mathsf{G})$, $\{\neg\mathsf{G}\}\vdash\neg(\mathsf{F}{\wedge}\mathsf{G})$;
(*iii*) $\{\mathsf{F}\}\vdash\mathsf{F}{\vee}\mathsf{G}$, $\{\mathsf{G}\}\vdash\mathsf{F}{\vee}\mathsf{G}$, $\{\neg\mathsf{F},\neg\mathsf{G}\}\vdash\neg(\mathsf{F}{\vee}\mathsf{G})$;
(*iv*) $\{\neg\mathsf{F}\}\vdash\mathsf{F}{\Rightarrow}\mathsf{G}$, $\{\mathsf{G}\}\vdash\mathsf{F}{\Rightarrow}\mathsf{G}$, $\{\mathsf{F},\neg\mathsf{G}\}\vdash\neg(\mathsf{F}{\Rightarrow}\mathsf{G})$.

DÉMONSTRATION. (*i*) La formule $\mathsf{F}{\Rightarrow}\neg\neg\mathsf{F}$ est une instance de l'axiome A_3, d'où le résultat par le théorème de la déduction.

(*ii*) La formule $\mathsf{F}{\Rightarrow}(\mathsf{G}{\Rightarrow}(\mathsf{F}{\wedge}\mathsf{G}))$ est une instance de l'axiome A_6; par le théorème de la déduction, on déduit $\{\mathsf{F}\}\vdash\mathsf{G}{\Rightarrow}(\mathsf{F}{\wedge}\mathsf{G})$, puis $\{\mathsf{F},\mathsf{G}\}\vdash\mathsf{F}{\wedge}\mathsf{G}$. La formule $(\mathsf{F}{\wedge}\mathsf{G}){\Rightarrow}\mathsf{F}$ est une instance

de l'axiome A_7; par le théorème de la déduction, on obtient $\{\mathsf{F}\wedge\mathsf{G}\}\vdash\mathsf{F}$, d'où, par le lemme 3.8(i), $\{\neg\mathsf{F}\}\vdash\neg(\mathsf{F}\wedge\mathsf{G})$. L'argument est similaire pour $\{\neg\mathsf{G}\}\vdash\neg(\mathsf{F}\wedge\mathsf{G})$ en utilisant l'axiome A_8.

(iii) La formule $\mathsf{F}\Rightarrow(\mathsf{F}\vee\mathsf{G})$ est une instance de l'axiome A_9; par le théorème de la déduction, on déduit $\{\mathsf{F}\}\vdash\mathsf{F}\vee\mathsf{G}$; l'argument est similaire pour $\{\mathsf{G}\}\vdash\mathsf{F}\vee\mathsf{G}$ en partant de l'axiome A_{10}. Ensuite, la formule $\neg\mathsf{F}\Rightarrow((\mathsf{F}\vee\mathsf{G})\Rightarrow\mathsf{G})$ est une instance de l'axiome A_{11}; par le théorème de la déduction, on obtient $\{\neg\mathsf{F}\}\vdash(\mathsf{F}\vee\mathsf{G})\Rightarrow\mathsf{G}$, puis $\{\neg\mathsf{F},\mathsf{F}\vee\mathsf{G}\}\vdash\mathsf{G}$, et, de là, $\{\neg\mathsf{F},\neg\mathsf{G}\}\vdash\neg(\mathsf{F}\vee\mathsf{G})$ en utilisant le lemme 3.8(i).

(iv) Par le lemme 3.8(ii), on a $\{\mathsf{F},\neg\mathsf{F}\}\vdash\mathsf{G}$, d'où, par le théorème de la déduction, $\{\neg\mathsf{F}\}\vdash\mathsf{F}\Rightarrow\mathsf{G}$. Ensuite, $\mathsf{G}\Rightarrow(\mathsf{F}\Rightarrow\mathsf{G})$ est une instance de l'axiome A_1, et on déduit $\{\mathsf{G}\}\vdash\mathsf{F}\Rightarrow\mathsf{G}$ par le théorème de la déduction. Enfin, $\{\mathsf{F},\mathsf{F}\Rightarrow\mathsf{G}\}\vdash\mathsf{G}$ est vrai par définition de la règle de coupure, et on en déduit $\{\mathsf{F},\neg\mathsf{G}\}\vdash\neg(\mathsf{F}\Rightarrow\mathsf{G})$ par le lemme 3.8(i). □

Le résultat suivant établit une première relation entre une hypothèse sémantique (mettant en jeu la satisfaction) et une conclusion syntaxique (mettant en jeu une preuve).

LEMME 3.10. *Pour V fonction partielle de $\{\boldsymbol{X}_\mathsf{i}\,;\,\mathsf{i}\geqslant 1\}$ dans $\{0,1\}$, on pose*

$$\mathsf{T}_V=\{\boldsymbol{X}_\mathsf{i}\,;\,V(\boldsymbol{X}_\mathsf{i})=1\}\cup\{\neg\boldsymbol{X}_j\,;\,V(\boldsymbol{X}_j)=0\}. \tag{3.1}$$

Alors, pour toute formule H telle que $\mathsf{Var}(\mathsf{H})$ soit inclus dans le domaine de V, la relation $V\models\mathsf{H}$ entraîne $\mathsf{T}_V\vdash\mathsf{H}$, et la relation $V\not\models\mathsf{H}$ entraîne $\mathsf{T}_V\vdash\neg\mathsf{H}$.

(Les implications réciproques sont triviales : par construction, on a $V\models\mathsf{T}_V$, et donc $\mathsf{T}_V\vdash\mathsf{F}$ entraîne $V\models\mathsf{F}$.)

DÉMONSTRATION. Nous utilisons une induction sur la complexité de H. Si H est une variable, alors le résultat est vrai par définition de T_V.

Supposons $\mathsf{H}=\neg\mathsf{F}$. L'hypothèse $V\models\mathsf{H}$ entraîne $V\not\models\mathsf{F}$, d'où, par hypothèse d'induction, $\mathsf{T}_V\vdash\neg\mathsf{F}$, c'est-à-dire $\mathsf{T}_V\vdash\mathsf{H}$. A l'opposé, l'hypothèse $V\not\models\mathsf{H}$ entraîne $V\models\mathsf{F}$, d'où $\mathsf{T}_V\vdash\mathsf{F}$ par hypothèse d'induction, et, enfin $\mathsf{T}_V\vdash\neg\neg\mathsf{F}$, c'est-à-dire $\mathsf{T}_V\vdash\neg\mathsf{H}$, puisqu'on a $\{\mathsf{F}\}\vdash\neg\neg\mathsf{F}$ par le lemme 3.9(i).

Supposons $\mathsf{H}=\mathsf{F}\wedge\mathsf{G}$. L'hypothèse $V\models\mathsf{H}$ entraîne $V\models\mathsf{F}$ et $V\models\mathsf{G}$, d'où $\mathsf{T}_V\vdash\mathsf{F}$ et $\mathsf{T}_V\vdash\mathsf{G}$ par hypothèse d'induction, puis $\mathsf{T}_V\vdash\mathsf{F}\wedge\mathsf{G}$, c'est-à-dire $\mathsf{T}_V\vdash\mathsf{H}$, puisqu'on a $\{\mathsf{F},\mathsf{G}\}\vdash\mathsf{F}\wedge\mathsf{G}$ par le lemme 3.9(ii). A l'opposé, l'hypothèse $V\not\models\mathsf{H}$ entraîne $V\not\models\mathsf{F}$ ou $V\not\models\mathsf{G}$, d'où $\mathsf{T}_V\vdash\neg\mathsf{F}$ ou $\mathsf{T}_V\vdash\neg\mathsf{G}$ par hypothèse d'induction, et, enfin $\mathsf{T}_V\vdash\neg(\mathsf{F}\wedge\mathsf{G})$, c'est-à-dire $\mathsf{T}_V\vdash\neg\mathsf{H}$, puisqu'on a $\{\mathsf{F}\}\vdash\neg(\mathsf{F}\wedge\mathsf{G})$ et $\{\mathsf{G}\}\vdash\neg(\mathsf{F}\wedge\mathsf{G})$ par le lemme 3.9(ii).

Supposons $\mathsf{H}=\mathsf{F}\vee\mathsf{G}$. L'hypothèse $V\models\mathsf{H}$ entraîne $V\models\mathsf{F}$ ou $V\models\mathsf{G}$, d'où $\mathsf{T}_V\vdash\mathsf{F}$ ou $\mathsf{T}_V\vdash\mathsf{G}$ par hypothèse d'induction, puis $\mathsf{T}_V\vdash\mathsf{F}\vee\mathsf{G}$, c'est-à-dire $\mathsf{T}_V\vdash\mathsf{H}$, puisqu'on a $\{\mathsf{F}\}\vdash\mathsf{F}\vee\mathsf{G}$ et $\{\mathsf{G}\}\vdash\mathsf{F}\vee\mathsf{G}$ par le lemme 3.9(iii). A l'opposé, l'hypothèse $V\not\models\mathsf{H}$ entraîne $V\not\models\mathsf{F}$ et $V\not\models\mathsf{G}$, d'où $\mathsf{T}_V\vdash\neg\mathsf{F}$ et $\mathsf{T}_V\vdash\neg\mathsf{G}$ par hypothèse d'induction, et, enfin $\mathsf{T}_V\vdash\neg(\mathsf{F}\vee\mathsf{G})$, c'est-à-dire $\mathsf{T}_V\vdash\neg\mathsf{H}$, puisqu'on a $\{\neg\mathsf{F},\neg\mathsf{G}\}\vdash\neg(\mathsf{F}\vee\mathsf{G})$ par le lemme 3.9(iii).

Supposons enfin $\mathsf{H}=\mathsf{F}\Rightarrow\mathsf{G}$. L'hypothèse $V\models\mathsf{H}$ entraîne $V\not\models\mathsf{F}$ ou $V\models\mathsf{G}$, d'où $\mathsf{T}_V\vdash\neg\mathsf{F}$ ou $\mathsf{T}_V\vdash\mathsf{G}$ par hypothèse d'induction, puis $\mathsf{T}_V\vdash\mathsf{F}\Rightarrow\mathsf{G}$, c'est-à-dire $\mathsf{T}_V\vdash\mathsf{H}$, puisqu'on a $\{\neg\mathsf{F}\}\vdash\mathsf{F}\Rightarrow\mathsf{G}$ et $\{\mathsf{G}\}\vdash\mathsf{F}\Rightarrow\mathsf{G}$ par le lemme 3.9(iv). A l'opposé, l'hypothèse $V\not\models\mathsf{H}$ entraîne $V\models\mathsf{F}$ et $V\not\models\mathsf{G}$, d'où $\mathsf{T}_V\vdash\mathsf{F}$ et $\mathsf{T}_V\vdash\neg\mathsf{G}$ par hypothèse d'induction, et, enfin $\mathsf{T}_V\vdash\neg(\mathsf{F}\Rightarrow\mathsf{G})$, c'est-à-dire $\mathsf{T}_V\vdash\neg\mathsf{H}$, puisqu'on a $\{\mathsf{F},\neg\mathsf{G}\}\vdash\neg(\mathsf{F}\Rightarrow\mathsf{G})$ par le lemme 3.9(iv). □

On peut maintenant démontrer le théorème de complétude.

Démonstration de la proposition 3.7. Soit F une formule valide, et soit $\boldsymbol{X}_\mathsf{n}$ la variable de plus grand indice figurant dans F. On montre par récurrence sur k décroissant de n à 0 que, quelle que soit l'application $V : \{\boldsymbol{X}_1, \dots, \boldsymbol{X}_\mathsf{k}\} \to \{0,1\}$, on a $\mathsf{T}_V \vdash \mathsf{F}$.

Pour $\mathsf{k} = \mathsf{n}$, l'application V attribue une valeur à chaque variable figurant dans F, et, par hypothèse, on a $V \models \mathsf{F}$, donc, par le lemme 3.10, on a $\mathsf{T}_V \vdash \mathsf{F}$.

Supposons $\mathsf{k} < \mathsf{n}$, et soit V une application définie sur $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_\mathsf{k}\}$. Notons V_1 et V_0 les deux applications définies sur $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_{\mathsf{k}+1}\}$ obtenues en complétant V respectivement par 1 et 0 en $\boldsymbol{X}_{\mathsf{k}+1}$. L'hypothèse de récurrence affirme que F est prouvable à partir de T_{V_1} et de T_{V_0}. Or, par construction, on a $\mathsf{T}_{V_1} = \mathsf{T}_V \cup \{\boldsymbol{X}_\mathsf{k}\}$ et $\mathsf{T}_{V_0} = \mathsf{T}_V \cup \{\neg\boldsymbol{X}_\mathsf{k}\}$, et donc F est prouvable à partir de $\mathsf{T}_V \cup \{\boldsymbol{X}_\mathsf{k}\}$ et de $\mathsf{T}_V \cup \{\neg\boldsymbol{X}_\mathsf{k}\}$. Le lemme 3.8(*iii*) entraîne alors que F est prouvable à partir de T_V.

Pour $\mathsf{k} = 0$, on obtient que F est prouvable à partir d'un ensemble d'hypothèses vide. □

▷ *Les ingrédients principaux de la démonstration précédente sont la possibilité d'utiliser un raisonnement par cas — qui suppose le principe du tiers-exclu, ainsi que la possibilité d'utiliser les hypothèses autant de fois qu'on veut — et le lemme 3.10, qui fait le lien entre sémantique et syntaxe. La démonstration de ce dernier repose sur les formules du lemme 3.9, lesquelles à leur tour reposent sur le choix des axiomes; celui-ci n'est pas vraiment important, l'essentiel est de pouvoir établir une réserve de relations de prouvabilité suffisante pour monter l'induction sur les formules du lemme 3.10. Noter que, si on restreint la syntaxe aux connecteurs $\Rightarrow$ et $\neg$, alors il suffit d'introduire les axiomes A_1 à A_5, et les vérifications à effectuer sont grandement raccourcies.*

Le théorème de complétude montre que, pour autant qu'on se contente de la sémantique booléenne du tout ou rien et qu'on n'exige rien de plus de l'implication que ce que contient la forme « non F ou G *», les schémas simples que constituent la règle de coupure et les axiomes de la logique booléenne épuisent les possibilités du raisonnement propositionnel.* ◁

3.4. Extensions et applications.

▶ On établit diverses extensions et corollaires du théorème de complétude de la logique booléenne $\mathcal{L}_\bullet$, en commençant par une forme globale de complétude, qui, au lieu de concerner une seule formule, met en jeu un ensemble quelconque de formules. ◀

Définition 3.11. (consistant) Un ensemble de formules T est dit *consistant* s'il n'existe pas de formule H telle qu'on ait à la fois $\mathsf{T} \vdash \mathsf{H}$ et $\mathsf{T} \vdash \neg\mathsf{H}$.

Dire que T est consistant signifie donc qu'on ne peut pas prouver à partir de T une chose et son contraire.

Lemme 3.12. (*i*) *Si* T *est un ensemble consistant de formules propositionnelles, et si* F *est une formule quelconque, alors l'un au moins des ensembles* $\mathsf{T} \cup \{\mathsf{F}\}$, $\mathsf{T} \cup \{\neg\mathsf{F}\}$ *est consistant.*

(*ii*) *Un ensemble de formules propositionnelles est consistant si et seulement tous ses sous-ensembles finis sont consistants.*

(*iii*) *Toute réunion filtrante*[4] *(donc en particulier toute réunion croissante) d'ensembles de formules propositionnelles consistants est consistante.*

[4] c'est-à-dire telle que deux éléments quelconques de l'ensemble d'indices ont toujours un majorant commun

DÉMONSTRATION. (*i*) Le lemme 3.8(*ii*) montre que, si T n'est pas consistant, alors T prouve toute formule G. Supposons que $\mathsf{T}\cup\{\mathsf{F}\}$ et $\mathsf{T}\cup\{\neg\mathsf{F}\}$ sont non consistants. Alors, d'après ce qui précède, pour toute formule H, à la fois $\mathsf{T}\cup\{\mathsf{F}\}$ et $\mathsf{T}\cup\{\neg\mathsf{F}\}$ prouvent H et $\neg\mathsf{H}$, donc, par le lemme 3.8(*iii*), T prouve H et $\neg\mathsf{H}$, donc T est non consistant.

(*ii*) Supposons T non consistant. Il existe donc une formule H, une preuve $\mathsf{F}_1,...,\mathsf{F}_\mathsf{p}$ de H à partir de T, et une preuve $\mathsf{G}_1,...,\mathsf{G}_\mathsf{q}$ de $\neg\mathsf{H}$ à partir de T. Soit T_0 le sous-ensemble de T constitué par les formules de T apparaissant parmi $\mathsf{F}_1,...,\mathsf{F}_\mathsf{p},\mathsf{G}_1,...,\mathsf{G}_\mathsf{q}$. Alors T_0 est fini, et, par construction, les preuves $\mathsf{F}_1,...,\mathsf{F}_\mathsf{p}$ et $\mathsf{G}_1,...,\mathsf{G}_\mathsf{q}$ sont des preuves à partir de T_0. Donc T_0 n'est pas consistant. Par conséquent, tout ensemble non consistant possède un sous-ensemble fini non consistant. Inversement, il est clair tout sous-ensemble (fini ou non) d'un ensemble consistant est consistant

(*iii*) Soit $\mathsf{T}=\bigcup_{i\in I}\mathsf{T}_i$, où $(I,<)$ est un ensemble ordonné filtrant supérieurement et où $i<j$ entraîne $\mathsf{T}_i\subseteq\mathsf{T}_j$. Supposons T non consistante. Comme dans (*ii*), il existe donc une formule H, une preuve $\mathsf{F}_1,...,\mathsf{F}_\mathsf{p}$ de H à partir de T, et une preuve $\mathsf{G}_1,...,\mathsf{G}_\mathsf{q}$ de $\neg\mathsf{H}$ à partir de T. Chaque formule F_k ou G_k appartient à un ensemble $\mathsf{T}_{i_\mathsf{k}}$, donc à T_i, où i est un majorant commun de tous les indices i_k intervenant (qui sont en nombre fini). On a alors $\mathsf{T}_i\vdash\mathsf{H}$ et $\mathsf{T}_i\vdash\neg\mathsf{H}$, donc T_i n'est pas consistante. □

PROPOSITION 3.13. (théorème de complétude, forme globale) *Tout ensemble consistant de formules propositionnelles est satisfaisable.*

DÉMONSTRATION. Soit T un ensemble consistant de formules propositionnelles. On définit par récurrence une suite d'ensembles T_n en posant $\mathsf{T}_0=\mathsf{T}$ puis, pour tout n,

$$\mathsf{T}_{\mathsf{n}+1}=\begin{cases}\mathsf{T}_\mathsf{n}\cup\{\boldsymbol{X}_\mathsf{n}\} & \text{si } \mathsf{T}_\mathsf{n}\cup\{\boldsymbol{X}_\mathsf{n}\} \text{ est consistant,}\\ \mathsf{T}_\mathsf{n}\cup\{\neg\boldsymbol{X}_\mathsf{n}\} & \text{sinon.}\end{cases}$$

On pose enfin $\mathsf{T}_\infty=\bigcup_{\mathsf{n}\in\mathbb{N}}\mathsf{T}_\mathsf{n}$. Par le lemme 3.12(*i*), on montre par récurrence sur n que T_n est consistant pour tout n. Alors, par le lemme 3.12(*iii*), T_∞ est consistant.

On définit alors une affectation V en posant $V(\boldsymbol{X}_\mathsf{i})=1$ pour $\boldsymbol{X}_\mathsf{i}\in\mathsf{T}_\infty$, et $V(\boldsymbol{X}_\mathsf{i})=0$ sinon, c'est-à-dire pour $\neg\boldsymbol{X}_\mathsf{i}\in\mathsf{T}_\infty$. Supposons F non satisfaite par V. Alors, par le lemme 3.10, on a $\mathsf{T}_V\vdash\neg\mathsf{F}$, donc $\mathsf{T}_\infty\vdash\neg\mathsf{F}$ puisque, par définition, T_∞ inclut T_V. Comme T_∞ est consistant, on déduit $\mathsf{T}_\infty\nvdash\mathsf{F}$, donc en particulier $\mathsf{F}\notin\mathsf{T}$. Par conséquent, $\mathsf{F}\in\mathsf{T}$ entraîne $V\models\mathsf{F}$: l'affection V satisfait T. □

▷ *Qu'il s'agisse de la forme locale ou globale du théorème de complétude, l'ingrédient technique essentiel est le lemme 3.10. Noter que la forme locale (proposition 3.7) se déduit aisément de la forme globale (proposition 3.13) : si* F *est valide, alors* $\{\neg\mathsf{F}\}$ *est non satisfaisable, donc non consistant, et il existe donc* H *tel qu'on ait à la fois* $\{\neg\mathsf{F}\}\vdash\mathsf{H}$ *et* $\{\neg\mathsf{F}\}\vdash\neg\mathsf{H}$. *Par le lemme 3.8(ii), on déduit* $\{\neg\mathsf{F}\}\vdash\mathsf{F}$, *d'où* $\vdash\mathsf{F}$ *par le lemme 3.8(iii) puisqu'on a* $\{\mathsf{F}\}\vdash\mathsf{F}$.

A l'inverse, la forme globale se déduit de la forme locale dans le cas d'un ensemble T *fini. En effet, supposons* $\mathsf{T}=\{\mathsf{F}_1,\dots,\mathsf{F}_\mathsf{n}\}$. *Si* T *n'est pas satisfaisable, la formule* $\neg(\mathsf{F}_1\wedge\cdots\wedge\mathsf{F}_\mathsf{n})$ *est valide, donc prouvable. On a donc* $\{\neg\mathsf{H}\}\vdash\neg(\mathsf{F}_1\wedge\cdots\wedge\mathsf{F}_\mathsf{n})$ *pour toute formule* H, *donc, par le lemme 3.8(i),* $\{\mathsf{F}_1\wedge\cdots\wedge\mathsf{F}_\mathsf{n}\}\vdash\mathsf{H}$, *d'où* $\mathsf{T}\vdash\mathsf{H}$. *Appliquant ceci à une formule* F *puis à sa négation* $\neg\mathsf{F}$, *on déduit que* T *est non consistante.*

Si on considère une version plus générale de la logique $\mathcal{L}_\bullet$ *dans laquelle les variables propositionnelles sont indexées par un ensemble quelconque* I, *alors une forme d'axiome du choix peut être nécessaire pour étendre la proposition 3.13, soit en utilisant la même démonstration une fois les variables* $\boldsymbol{X}_\mathsf{i}$ *bien ordonnées, soit en arguant du lemme de Zorn pour affirmer directement l'existence d'un ensemble consistant maximal* T' *incluant l'ensemble* T *initial et en montrant que, pour chaque variable* $\boldsymbol{X}_\mathsf{i}$, *un tel ensemble maximal doit contenir soit* $\boldsymbol{X}_\mathsf{i}$, *soit* $\neg\boldsymbol{X}_\mathsf{i}$. *La propriété que les ensembles consistants de formules ordonnés par inclusion forment un ensemble ordonné inductif résulte immédiatement du lemme 3.12(iii).* ◁

COROLLAIRE 3.14. *Pour tout ensemble* T *de formules propositionnelles, et toute formule propositionnelle* F*, il y a équivalence entre* $\mathsf{T} \vdash \mathsf{F}$ *et* $\mathsf{T} \models \mathsf{F}$*.*

DÉMONSTRATION. Si V est une affectation satisfaisant T, elle satisfait aussi les axiomes, qui sont valides, et donc, de là, toute formule prouvable à partir de T et des axiomes. Par conséquent, $\mathsf{T} \vdash \mathsf{F}$ entraîne $\mathsf{T} \models \mathsf{F}$.

Inversement, supposons $\mathsf{T} \models \mathsf{F}$. Ceci signifie que l'ensemble $\mathsf{T} \cup \{\neg\mathsf{F}\}$ n'est pas satisfaisable. Par le théorème de complétude, il n'est donc pas consistant, et il existe donc un sous-ensemble fini $\{\mathsf{F}_1, \dots, \mathsf{F}_\mathsf{n}\}$ de T tel que $\{\mathsf{F}_1, \dots, \mathsf{F}_\mathsf{n}, \neg\mathsf{F}\}$ n'est pas consistant, donc n'est pas satisfaisable. C'est dire que la formule $\mathsf{F}_1 \Rightarrow (\mathsf{F}_2 \Rightarrow \dots (\mathsf{F}_\mathsf{n} \Rightarrow \mathsf{F}) \dots)$ est valide. Par le théorème de complétude, cette formule est donc prouvable, c'est-à-dire qu'on a $\vdash \mathsf{F}_1 \Rightarrow (\mathsf{F}_2 \Rightarrow \dots (\mathsf{F}_\mathsf{n} \Rightarrow \mathsf{F}) \dots)$, d'où $\{\mathsf{F}_1, \dots, \mathsf{F}_\mathsf{n}\} \vdash \mathsf{F}$ en appliquant n fois la coupure. □

Une autre application est le résultat suivant, appelé *théorème de compacité* pour la logique booléenne :

COROLLAIRE 3.15. *Soit* T *un ensemble de formules propositionnelles. Alors* T *est satisfaisable si et seulement si tout sous-ensemble fini de* T *l'est.*

DÉMONSTRATION. Il est clair que la condition est nécessaire. Réciproquement, supposons tout sous-ensemble fini de T satisfaisable. Alors tout sous-ensemble fini de T est consistant, donc, par le lemme 3.12(*ii*), T lui-même est consistant, et, par le théorème de complétude, T est satisfaisable. □

▷ *On peut mentionner une autre démonstration, fondée sur le théorème de Tychonoff. L'ensemble de toutes les affectations est l'espace de Cantor* $\{0,1\}^{\mathbb{N}}$*. On le munit de la topologie-produit de la topologie discrète sur* $\{0,1\}$*, engendrée par des ouverts-fermés de la forme*

$$U_{\varepsilon_0,\dots,\varepsilon_k} = \{V\,;\, V(0) = \varepsilon_0, \dots, V(k) = \varepsilon_k\}.$$

Comme $\{0,1\}$ *est compact et que tout produit d'espaces compacts est compact,* $\{0,1\}^{\mathbb{N}}$ *est compact. Pour* F *formule propositionnelle, soit* $S(\mathsf{F})$ *l'ensemble des affectations satisfaisant* F*. Comme il n'existe qu'un nombre fini de variables apparaissant dans* F*, l'ensemble* $S(\mathsf{F})$ *est une réunion finie d'ouverts-fermés, donc est lui-même un ouvert-fermé. Soit* T *un ensemble de formules propositionnelles. Dire que* T *est satisfaisable signifie que* $\bigcap_{\mathsf{F}\in T} S(\mathsf{F})$ *est non vide. Comme* $\{0,1\}^{\mathbb{N}}$ *est compact, si cette intersection de fermés est vide, il existe un sous-ensemble fini* T_0 *de* T *tel que l'intersection* $\bigcap_{\mathsf{F}\in\mathsf{T}_0} S(\mathsf{F})$ *est vide, c'est-à-dire tel que* T_0 *est non satisfaisable.* ◁

4. Pouvoir d'expression de la logique booléenne

► On montre que de nombreux problèmes peuvent se coder en la question de la satisfaisabilité d'un ensemble de formules propositionnelles, et on en déduit le théorème de Cook et Levin sur le caractère NP-complet du problème SAT. ◄

▷ *On pourrait avoir le sentiment que la logique booléenne est rudimentaire et ne peut exprimer que des relations triviales du genre* $\mathsf{F} \Rightarrow (\mathsf{F} \Rightarrow \mathsf{F})$*. Ce point de vue serait erroné, car le pouvoir d'expression de la logique booléenne est grand : de nombreux problèmes peuvent se ramener au problème de déterminer si une famille de formules propositionnelles est valide, ou à trouver au moins une affectation la satisfaisant. C'est en particulier le cas pour des problèmes où l'espace des configurations est discret : une configuration peut alors être codée par une affectation de valeurs à des variables propositionnelles, et, dans de nombreux cas, les contraintes du système peuvent être traduites par des formules propositionnelles.* ◁

4.1. Un exemple.

► On montre sur un exemple comment un problème concernant des nombres entiers peut être codé en un problème de satisfaisabilité. ◄

▷ *Considérons le jeu*

$$\begin{array}{rcccccc} & M & A & N & G & E & R \\ + & M & A & N & G & E & R \\ \hline G & R & O & S & S & I & R \end{array}$$

Il s'agit de trouver des chiffres distincts entre 0 *et* 8 *à substituer aux lettres de sorte que l'addition soit correcte. A priori, ce problème est un problème de chiffres et d'arithmétique, pas de logique. Mais il est aisé de coder la question à l'aide de variables propositionnelles. Introduisons en effet 9 variables notées* $[M = 0]$, $[M = 1]$, ... , $[M = 8]$, *et de même pour les huit autres lettres* A, N, G, E, R, O, S, I *(donc 81 variables en tout). Le choix d'une valeur entre* 0 *et* 8, *par exemple* 4, *pour le chiffre* M *correspond à l'affectation de la valeur* 1 *à la variable* $[M = 4]$, *et de la valeur* 0 *pour les variables* $[M = k]$ *avec* $k \neq 4$. *Inversement, une affectation de valeurs pour les variables* $[M = 0]$, *etc. détermine, dans les bons cas, une valeur pour* M. *Le point important est le fait que toutes les contraintes issues du problème initial peuvent se traduire en la satisfaction d'un certain nombre de formules propositionnelles portant sur les 81 variables* $[M = 0]$, $[M = 1]$, ... *La première contrainte est que chaque lettre, par exemple* M, *doit avoir une et une seule valeur: or ceci se traduit par la satisfaction de la formule*

$$\bigvee_{i=0}^{i=8} [M = i] \wedge \bigwedge_{i=0}^{i=8} \Big([M = i] \Rightarrow (\bigwedge_{j \neq i} \neg [M = j]) \Big)\ ^{5}.$$

De même pour chacune des lettres A, ... , I. *Il est facile de traduire la contrainte que les valeurs de deux lettres différentes sont distinctes. Reste à traduire le problème lui-même, c'est-à-dire la correction de l'addition. Or, ceci à son tour s'exprime par la satisfaction d'un nombre fini de formules propositionnelles, telles que*

$$\bigwedge_{0 \leqslant i \leqslant 8} ([R = i]) \Rightarrow [R = 2i \bmod 10]),$$

et de même le chiffre des dizaines, à ceci près qu'il sera commode d'introduire des variables supplémentaires exprimant la présence ou l'absence de retenue sur chacune des colonnes. Finalement, en prenant la conjonction de toutes les formules précédentes, on obtient une (grosse) formule propositionnelle F *telle que la détermination d'une affectation satisfaisant* F *équivaut à la solution du problème initial* [6]. ◁

4.2. Machines de Turing.

► Afin de pouvoir énoncer le théorème de Cook et Levin dans la section suivante, on introduit brièvement le contexte des machines de Turing et la notion de complexité algorithmique. ◄

▷ *Tout développement mathématique sur les questions d'effectivité requiert le choix d'un modèle de calcul, afin de donner un sens précis à la notion de problème décidable (ou résoluble) et de complexité algorithmique. Il existe de bons arguments pour choisir comme modèle de calcul les machines de Turing, qui sont des sortes de calculettes rudimentaires.*

[5] où les opérateurs $\bigvee$ et $\bigwedge$ sont utilisés pour $\vee$ et $\wedge$ de la même façon qu'on utilise $\sum$ pour $+$

[6] D'après la proposition 2.8, on sait qu'il existe un algorithme décidant la satisfaisabilité d'une formule propositionnelle F, à savoir l'essai de toutes les affectations possibles à la famille finie des variables présentes dans F, et on peut ainsi, au moins de façon théorique, résoudre le problème. Des problèmes tels que celui décrit ici sont classiquement utilisés pour tester les algorithmes pratiques; évidemment, dans le cas présent, on peut aussi résoudre le problème à la main et obtenir l'unique solution $536140 + 536140 = 1072280$.

Une machine de Turing M est un programme fini spécifiant une famille de transitions légales sur un ensemble (infini) de configurations, et un calcul de M est une suite (finie ou infinie) de configurations $(c_0, c_1, \dots)$ telle que (c_n, c_{n+1}) soit une transtion légale de M pour tout n. Dans la version la plus simple, une configuration peut être vue comme une suite de cases indexées par des entiers et contenant chacune au plus une lettre prise dans un certain alphabet Σ, plus la spécification d'une case particulière dite accessible, plus celle d'un état pris dans un ensemble fini Q correspondant intuitivement au contenu de la mémoire de M : une telle configuration est donc codée par un triplet (f, n, q) dans $(\Sigma \cup \{\Box\})^{\mathbb{Z}} \times \mathbb{Z} \times Q$, où on utilise $\Box$ pour représenter le blanc (absence de caractère). On considère une seule sorte de transition, consistant à lire le caractère s de la case accessible n, à remplacer celui-ci par un nouveau caractère s', rendre accessible une nouvelle case n' avec $n' = n \pm 1$, et passer dans un nouvel état q', les nouvelles valeurs ne dépendant que de s et de q. Autrement dit, une transition consiste à passer de (f, n, q) à (f', n', q') avec $f'(x) = f(x)$ pour $x \neq n$, et $n' = n + d$ avec $d = \pm 1$, et $f'(n)$, $n' - n$, et q' dépendant seulement de $f(n)$ et de q. La machine de Turing M est le programme spécifiant les transitions légales, par exemple sous la forme d'une liste finie de quintuplets (s, q, s', q', d) correspondant à « ancien caractère, ancien état, nouveau caractère, nouvel état, déplacement ». Si, pour chaque valeur de (s, q), il existe au plus une transition légale à partir de (s, q), on dit que la machine M est déterministe, et ceci correspond au fait qu'un seul calcul part d'une configuration donnée.

On considère des problèmes dits de décision : pour un ensemble de mots L sur un alphabet Σ, le problème de décision pour L consiste, étant donné un mot quelconque w sur Σ, à déterminer si w est ou non dans L. Moyennant des codages ad hoc, de très nombreux problèmes mathématiques peuvent être exprimés comme des problèmes de décision.

Dans ce contexte, on peut utiliser les machines de Turing pour résoudre les problèmes de décision comme suit. D'abord, on considère des machines dont l'ensemble d'états contient deux états spéciaux, « OUI » et « NON », à partir desquels aucune transition n'est possible. On dit alors qu'une machine de Turing M décide L si, pour chaque mot w de L, il existe un calcul de M partant de la configuration où w seul est écrit sur les cases numérotées 1, 2, ... , où la case initialement accessible est la case 1, et où l'état initial est un certain état prescrit à l'avance, et menant à l'état OUI, *et si, pour chaque mot w non dans L, tout calcul de M partant de la configuration initiale décrite ci-dessus mène à l'état « NON ». On obtient ainsi la notion d'ensemble décidable par machine de Turing, ou MT-décidable.*

Chaque calcul d'une machine de Turing a une longueur bien définie, à savoir le nombre de configurations successives, et on peut raffiner la notion de MT-décidabilité en déclarant, pour T fonction de $\mathbb{N}$ dans $\mathbb{N}$, que M décide L en temps T si, pour chaque mot w de longueur n, les calculs (acceptants ou refusants) de M à partir de w requièrent au plus $T(n)$ étapes de calcul. Un cas particulier important en pratique est celui où la fonction T est polynomiale : un ensemble L est dit MT-décidable en temps polynomial s'il existe un polynôme T et une machine de Turing M tels que M décide L en temps T. ◁

DÉFINITION 4.1. (classes P et NP) Un ensemble L est dit de classe P (*resp.* NP) s'il existe une machine de Turing déterministe (*resp.* une machine de Turing) décidant L en temps polynomial.

▷ *Il est facile de se convaincre qu'un calcul de machine de Turing déterministe correspond au déroulement d'un algorithme, et de là que tout ensemble MT-décidable est décidable, au sens où il existe un algorithme qui le décide. L'implication réciproque, dont l'affirmation constitue ce qu'on appelle la thèse de Church, à savoir que tout algorithme peut être implémenté sur une machine de Turing, ne peut pas être démontrée, puisque la notion d'algorithme n'a pas été définie formellement. Par contre, aucun élément ne vient pour le moment la contredire, et, au contraire, la multiplicité des approches menant à la notion d'ensemble MT-décidable (voir chapitre VIII) va dans le sens de donner à celle-ci un caractère naturel et canonique. Il en est de même pour la version quantitative, c'est-à-dire lorsqu'on prend en compte la complexité des algorithmes.*

Moyennant ce qui précède, la classe P correspond à la classe des problèmes qui peuvent être résolus en temps polynomial, tandis que la classe NP correspond aux problèmes dont les solutions peuvent être prouvées en temps polynomial : considérer une machine de Turing non

déterministe équivaut à effectuer plusieurs calculs en parallèle, et un problème dans NP *est un problème qui peut être résolu en temps polynomial pourvu qu'on ait choisi les bonnes options. En d'autres termes, si on a deviné correctement les éléments de la solution, on peut prouver qu'on a bien une solution.* ◁

DÉFINITION 4.2. (SAT) On note SAT l'ensemble des formules propositionnelles satisfaisables en logique booléenne.

LEMME 4.3. *L'ensemble* SAT *est dans la classe* NP.

DÉMONSTRATION. On reprend la démonstration de la proposition 2.8. Si une formule F a, en tant que mot, la longueur n, l'évaluation de F en une affectation fixée requiert au plus $O(\mathsf{n}^2)$ étapes élémentaires de calcul (en un sens à préciser). Comme le nombre de variables dans F est certainement inférieur à n, on déduit que la satisfaisabilité de F peut se décider en $O(2^{c\mathsf{n}})$ étapes pour tout c plus grand que 1: l'ensemble SAT est donc décidable en temps exponentiel. Par contre, si on suppose qu'une formule F est satisfaisable et qu'on a deviné une affectation V satisfaisant F, alors on peut prouver que V satisfait F en $O(\mathsf{n}^2)$ étapes. A l'inverse, si F n'est pas satisfaisable, toute évaluation mène à un échec dans le même temps. Implémenter cette méthode sur une machine d Turing (avec les mêmes bornes de complexité) est facile. □

4.3. Le théorème de Cook et Levin.

► On esquisse la démonstration du théorème de Cook et Levin affirmant que toute solution déterministe polynomiale au problème SAT fournirait une solution de même complexité pour tout problème dans la classe NP. ◄

▷ *Sur le modèle de l'anecdotique exemple de la section 4.1, tout problème d'existence pouvant être codé par une suite finie de variables propositionnelles peut, en un sens précis, se réduire au problème de satisfaisabilité booléenne. C'est ce fait qu'exploite le théorème de Cook et Levin, qui est un résultat de borne inférieure de complexité pour* SAT. ◁

DÉFINITION 4.4. (réduction) Soient Σ, Σ' deux alphabets, et L, L' deux ensembles de mots d'alphabet Σ et Σ' respectivement. On dit qu'une application F de l'ensemble des mots sur Σ dans l'ensemble des mots sur Σ' est une *réduction polynomiale* de L à L' si

- la fonction F est calculable par machine de Turing en temps polynomial [7],
- pour tout mot w sur Σ, on a $w \in L$ si et seulement si on a $F(w) \in L'$.

PROPOSITION 4.5. (théorème de Cook et Levin) *Pour tout ensemble L appartenant à la classe* NP, *il existe une réduction polynomiale de L à* SAT.

DÉMONSTRATION. Le principe de la démonstration est le suivant. Supposons que L est un ensemble de mots sur l'alphabet Σ pour lequel il existe une procédure de décision non déterministe polynomiale. Par hypothèse, il existe une machine de Turing non déterministe M qui décide L en temps Cn^d. On suppose fixée une numérotation $\{x_1, \dots, x_r\}$ de l'alphabet de M, et, de même, une numératation $\{q_1, \dots, q_e\}$ de l'ensemble des états de M. On considèrera le blanc comme le caractère numéro 0 de l'alphabet.

[7] on dit qu'une fonction F est calculable par machine de Turing s'il existe une machine de Turing déterministe qui, pour tout mot w sur Σ, mène en un nombre fini d'étapes de la configuration initiale où w est écrit seul à la configuration finale où $F(w)$ est écrit seul

Soit w un mot de longueur n sur l'alphabet de L. Par hypothèse, w est dans L si et seulement si il existe une suite de configurations $(c_0, c_1, \ldots, c_t)$ telle que

- c_0 est la configuration initiale associée à w,
- pour chaque i, le passage de c_{i-1} à c_i obéit aux règles de transition spécifiées par M,
- c_t est une configuration acceptante, c'est-à-dire dont l'état est « OUI ».

De plus, toujours par hypothèse, on sait que, si w est dans L, alors il existe une suite de configurations comme ci-dessus de longueur majorée par Cn^d. Quitte à modifier la machine de Turing M pour qu'elle reste dans l'état acceptant plutôt que de s'arrêter, on peut donc supposer qu'on a $t = Cn^d$.

Notons alors que, partant d'une configuration où le premier caractère du mot w est accessible, on ne pourra en Cn^d étapes accéder qu'aux cases dont le numéro est compris entre $-Cn^d$ et $+Cn^d$. De ce fait, et c'est là le point essentiel, une description complète du calcul $(c_0, \ldots, c_{Cn^d})$ peut être faite en spécifiant des paramètres en nombre fini borné par un polynôme en n. Plus précisément, une telle description peut être faite en spécifiant les valeurs « vrai » ou « faux » d'une famille de variables proppositionnelles. Typiquement, on peut décrire le calcul en donnant les valeurs de vérité de variables propositionnelles telles que

- « le contenu de la case numéro m à l'étape i est le caractère numéro j de l'alphabet »,
- « l'état à l'étape i est l'état numéro k »,
- « la case accessible à l'étape i est la case numéro m ».

En faisant varier les paramètres m, i, j, et k respectivement de $-Cn^d$ à $+Cn^d$, de 0 à Cn^d, de 1 à r, et de 1 à e, on obtient une description complète d'un calcul de longueur au plus Cn^d de M. Nous pouvons fixer une numérotation pour les variables du type ci-dessus, par exemple décider que la variable représentant la contrainte « le contenu de la case numéro m à l'étape i est le caractère numéro j de l'alphabet » est $\boldsymbol{X}_N$, où N est l'entier dont le développement en base 10 (ou 3...) est $1^m01^i01^j$ pour $m \geqslant 0$ et $2^{-m}01^i01^j$ pour $m < 0$. Dans la suite, nous noterons simplement $y_{m,i,j}$ cette variable, et, de même, nous noterons $z_{i,k}$ et $t_{m,i}$ les variables associées de même aux contraintes « l'état à l'étape i est l'état numéro k » et « la case numéro m est accessible à l'étape i ».

A ce point, nous avons obtenu, pour chaque n un *codage* des calculs de M de longueur au plus Cn^d par une affectation de valeurs aux variables propositionnelles. Ce codage est injectif, l'affectation définit complètement le calcul. Par contre, il n'est pas surjectif: de nombreuses affectations ne codent aucun calcul. Par exemple, une affectation dans laquelle les deux variables $z_{3,2}$ et $z_{3,5}$, qui sont associées respectivement à

« l'état à l'étape 3 est l'état numéro 2 », et

« l'état à l'étape 3 est l'état numéro 5 »,

reçoivent toutes deux la valeur 1 ne peut coder un calcul, puisque, dans une configuration de machine de Turing, il n'y a qu'un état à chaque étape.

Le point essentiel est alors le suivant: pour chaque mot w, nous pouvons écrire une formule propositionnelle F_w exprimant l'ensemble des contraintes que doit satisfaire une affectation V pour coder un calcul acceptant de M de longueur Cn^d à partir de w. La formule F_w est une (longue) conjonction de formules dont chacune exprime une contrainte élémentaire. Ces contraintes sont de trois sortes:

- traduire les contraintes sur c_0, c'est-à-dire exprimer que R code un calcul partant de w,
- traduire les contraintes sur le passage de c_{i-1} à c_i, c'est-à-dire exprimer que R code un calcul respectant les règles de transition de M,
- traduire les contraintes sur c_{Cn^d}, c'est-à-dire exprimer que R code un calcul acceptant.

Pour la première famille, un fragment typique de la formule est la conjonction des formules

$$y_{m,0,0} \wedge \neg y_{m,0,1} \wedge \cdots \wedge \neg y_{m,0,d}$$

pour $m = -CN^d, \dots, -1, 0, n+1, \dots, CN^d$: si cette formule est satisfaite par l'affectation V, alors V code un calcul où les cases de numéro autres que $1, \dots, n$ de la configuration initiale sont vides. De même, supposant qu'on a $w = x_{j_1} \dots x_{j_n}$, un autre fragment de F_w est la conjonction des formules

$$y_{m,0,j_m} \wedge \neg y_{m,0,1} \wedge \dots \wedge \neg \widehat{y_{m,0,j_m}} \wedge \dots \wedge y_{m,0,r}$$

pour $1 \leqslant m \leqslant n$: si cette formule est satisfaite par l'affectation V, alors V code un calcul dont la configuration initiale où w est écrit sur les cases 1 à n. De même, le fragment

$$z_{0,1} \wedge \neg z_{0,2} \wedge \dots \wedge \neg z_{0,e}$$

traduit qu'à l'étape 0, l'état est 1 (supposé initial), et que c'est le seul état.

Les contraintes de la deuxième famille sont semblables. Si les transitions permises par la table de M à partir de (x_j, q_k) sont

$$(x_{j_1}, \delta_1, q_{k_1}), \dots, (x_{j_p}, \delta_1, q_{k_p}),$$

on écrira, pour chaque j, et chaque i, des formules du type

$$\begin{aligned}(y_{m,i,j} \wedge z_{i,k}) \Rightarrow ((y_{m,i+1,j_1} \wedge \bigwedge_{j \neq j_1} \neg y_{m,i+1,j} \wedge z_{i+1,k_1} \wedge \bigwedge_{k \neq k_1} \neg z_{i+1,k}) \\ \vee \dots \vee (y_{m,i+1,j_p} \wedge \bigwedge_{j \neq j_p} \neg y_{m,i+1,j} \wedge z_{i+1,k_p} \wedge \bigwedge_{k \neq k_p} \neg z_{i+1,k}))\end{aligned}$$

qui expriment que, partant de (x_j, q_k) à l'étape i, les seules possibilités à l'étape $i+1$ sont celles stipulées par M. Noter que la formule ci-dessus n'est pas complète, car elle ne prend pas en compte le déplacement de la case accessible.

Les contraintes de la troisième famille sont très simples à exprimer: supposant que l'état acceptant est q_e, la seule formule $z_{Cn^d,e}$ traduit le fait que la calcul est acceptant. Les détails complets sont fastidieux, mais le point essentiel est qu'on peut effectivement écrire une formule F_w du type souhaité, *et* que l'écriture de cette formule à partir de w et de M s'effectue — plus exactement, peut être effectuée par une machine de Turing — de façon uniforme et en temps polynomial par rapport à la longueur de w. L'application $\mathsf{F} : w \mapsto \mathsf{F}_w$ est alors la réduction cherchée. En effet, par construction, toute affectation satisfaisant F_w code un calcul acceptant de M à partir de w, donc la formule F_w est satisfaisable si et seulement si le mot w est accepté par M: c'est dire que F_w est dans SAT si et seulement si w est dans L. □

Le résultat précédent implique que le problème SAT est en un sens précis le plus difficile des problèmes de la classe NP, puisque, si on savait résoudre (déterministiquement) SAT en temps polynomial, alors on saurait *ipso facto* résoudre en temps polynomial tout problème de la classe NP. On traduit ce fait en introduisant la notion de problème NP-*complet*:

DÉFINITION 4.6. (C-complet) Si C est une classe de complexité[8], on dit que le problème (ou le langage) L est C-*complet* si L appartient à C et si tout problème dans C se réduit polynomialement à L.

Le théorème de Cook et Levin affirme donc que SAT est NP-complet.

QUESTION 4.7. *A-t-on* $\mathsf{P} = \mathsf{NP}$ *?*

[8]typiquement, la famille des problèmes résolubles en temps T au moyen d'un certain type de modèle de calcul; la définition présente référant à une réduction polynomiale n'est pertinente que lorsque T est au moins polynomiale, et doit être adaptée pour des complexités plus petites, par exemple linéaire ou quadratique

▷ *La question est ouverte, et c'est probablement la plus célèbre des questions ouvertes d'informatique théorique. En termes informels, il s'agit de savoir si on peut résoudre en temps polynomial tout problème pour lequel on peut décider en temps polynomial si un candidat-solution est effectivement solution — soit encore tout problème pouvant être résolu en temps polynomial une fois devinées suffisamment d'informations complémentaires.* ◁

Appendice : d'autres logiques propositionnelles

► Le choix des connecteurs et des règles de déduction de la logique booléenne ont été justifiés par le souhait de modéliser de façon simple le raisonnement naturel. Des analyses plus détaillées, ou prenant en compte d'autres aspects du raisonnement, mènent à d'autres logiques formelles. ◄

La logique intuitionniste.

► Introduite dans les années 1910 par L.E.J. Brouwer, la logique intuitionniste, ou constructiviste, s'efforce de mieux rendre compte de la nature intuitive de l'implication que la logique booléenne, dont elle est une extension. Les formules sont les mêmes qu'en logique classique, et la notion de preuve est analogue, à l'exception de l'axiome du tiers exclu ; par contre, les sémantiques sont plus compliquées, l'une des plus simples étant obtenue en termes topologiques. ◄

▷ *On a remarqué que la sémantique booléenne qui déclare vraie tout implication* $\mathsf{F}\Rightarrow\mathsf{G}$ *telle que* F *soit fausse ne rend qu'imparfaitement compte de ce qu'on entend intuitivement par déduction. La logique intuitionniste cherche à mieux modéliser le raisonnement, en ne considérant une implication* $\mathsf{F}\Rightarrow\mathsf{G}$ *comme valide que si on sait effectivement passer d'une preuve pour* F *à une preuve pour* G*. Les formules de la logique propositionnelle intuitionniste sont les mêmes que celles de la logique propositionnelle booléenne. Par contre, la notion de preuve est modifiée en ce que l'axiome* $\neg\neg\boldsymbol{X}_1\Rightarrow\boldsymbol{X}_1$ *est omis. Ce faisant, toute formule prouvable en logique intuitionniste est prouvable en logique booléenne, mais la réciproque est fausse : on peut montrer que* $\neg\neg\boldsymbol{X}_1\Rightarrow\boldsymbol{X}_1$ *n'est pas prouvable en logique intuitionniste — ni, ce qui revient au même, la formule* $\boldsymbol{X}_1\vee\neg\boldsymbol{X}_1$ *dite du tiers exclu.*

La construction d'une sémantique pour laquelle on ait à la fois cohérence et complétude est plus délicate qu'en logique booléenne. Une des possibilités est d'affecter aux variables propositionnelles des valeurs qui sont des ouverts de $\mathbb{R}^2$*, et d'interpréter la conjonction et la disjonction comme intersection et réunion, et la négation comme l'****intérieur*** *du complémentaire — et non le complémentaire comme une sémantique booléenne le suggérerait. Les formules valides sont celles dont la valeur est* $\mathbb{R}^2$ *pour toute affectation de valeurs aux variables. On conçoit que* $\boldsymbol{X}_1\vee\neg\boldsymbol{X}_1$ *n'est pas valide, puisque sa valeur n'inclut pas la frontière de la valeur de* $\boldsymbol{X}_1$*.*

La logique intuitionniste étend la logique booléenne, en ce que l'application $\mathsf{F}\mapsto\neg\neg\mathsf{F}$ *définit une représentation fidèle de la logique booléenne dans un fragment de la logique intuitionniste :* F *est valide en logique booléenne si et seulement si* $\neg\neg\mathsf{F}$ *l'est en logique intuitionniste.*

Il existe un célèbre isomorphisme, dit de Curry-Howard, entre les formules prouvables en logique intuitionniste et les termes typables du lambda-calcul. Ceci explique l'intérêt spécifique de la logique intuitionniste en informatique théorique. ◁

La logique linéaire.

► Introduite dans les années 1980 par J.Y. Girard, la logique linéaire est une autre extension de la logique booléenne. ◄

▷ *Le point de départ est ici de considérer les variables propositionnelles comme représentant davantage des ressources que des vérités. Ce point de vue mène à ne plus tenir pour équivalentes les formules* $\boldsymbol{X}_1$ *et* $\boldsymbol{X}_1 \wedge \boldsymbol{X}_1$ *: utiliser deux fois* $\boldsymbol{X}_1$ *n'est pas équivalent à l'utiliser une fois. On est alors conduit à distinguer deux formes pour les connecteurs de conjonction, de disjonction et d'implication, ainsi qu'à introduire un nouveau connecteur* ! *(« factorielle ») signifiant la répétition indéfinie d'une ressource. Comme dans le cas de la logique intuitionniste, une notion de preuve a été développée, ainsi que diverses sémantiques, en particulier basées sur la notion de jeu.*

La logique linéaire est une extension de la logique intuitionniste : l'application $\mathsf{F} \mapsto \mathsf{F}!$ *définit une représentation fidèle de la logique intuitionniste dans un fragment de la logique linéaire : les ressources inépuisables sont analogues aux vérités. A la différence de la logique intuitionniste, la logique linéaire préserve le caractère involutif (d'une certaine forme) de la négation :* $\neg\neg\mathsf{F}$ *équivaut à* F. ◁

Exercices

Exercice 1. (algèbre de Boole) Montrer que $\{0,1\}$ équipé de $\wedge$, $\vee$ et $\neg$ est une algèbre de Boole.

Exercice 2. (connecteur de Scheffer) Montrer que toute application de $\{0,1\}^n$ dans $\{0,1\}$ peut se définir à partir des opérations $\wedge$, $\vee$ et $\neg$. En déduire que toute application de $\{0,1\}^n$ dans $\{0,1\}$ peut se définir à partir de l'unique opération $\uparrow$ définie par $\boldsymbol{X}_1 \uparrow \boldsymbol{X}_2 = \neg\boldsymbol{X}_1 \wedge \neg\boldsymbol{X}_2$.

Exercice 3. (méthode de réduction) (*i*) On note $\mathcal{L}_\bullet^+$ la logique dont les formules sont celles de $\mathcal{L}_\bullet$, plus deux nouveaux symboles $\bot, \top$ pouvant être substitués aux variables, et dont la sémantique étend celle de $\mathcal{L}_\bullet$ en déclarant $\top$ valide et $\bot$ non satisfaisable. On appelle *réduction* l'application $\mathtt{red}$ récursivement définie sur les formules de $\mathcal{L}_\bullet^+$ par : $\mathtt{red}(\mathsf{F}) = \mathsf{F}$ si F est une variable, $\bot$, ou $\top$; $\mathtt{red}(\neg\mathsf{F}) = \neg\mathtt{red}(\mathsf{F})$; $\mathtt{red}(\neg\top) = \bot$; $\mathtt{red}(\neg\bot) = \top$; $\mathtt{red}(\mathsf{F}\vee\mathsf{G}) = \mathtt{red}(\mathsf{F})\vee\mathtt{red}(\mathsf{G})$; $\mathtt{red}(\mathsf{F}\vee\top) = \mathtt{red}(\top\vee\mathsf{F}) = \top$; $\mathtt{red}(\mathsf{F}\vee\bot) = \mathtt{red}(\bot\vee\mathsf{F}) = \mathtt{red}(\mathsf{F})$; $\mathtt{red}(\mathsf{F}\wedge\mathsf{G}) = \mathtt{red}(\mathsf{F})\wedge\mathtt{red}(\mathsf{G})$; $\mathtt{red}(\mathsf{F}\wedge\top) = \mathtt{red}(\top\wedge\mathsf{F}) = \mathtt{red}(\mathsf{F})$; $\mathtt{red}(\mathsf{F}\wedge\bot) = \mathtt{red}(\bot\wedge\mathsf{F}) = \bot$; $\mathtt{red}(\mathsf{F}\Rightarrow\mathsf{G}) = \mathtt{red}(\mathsf{F})\Rightarrow\mathtt{red}(\mathsf{G})$; $\mathtt{red}(\mathsf{F}\Rightarrow\top) = \mathtt{red}(\bot\Rightarrow\mathsf{F}) = \top$; $\mathtt{red}(\mathsf{F}\Rightarrow\bot) = \neg\mathtt{red}(\mathsf{F})$; $\mathtt{red}(\bot\Rightarrow\mathsf{F}) = \mathtt{red}(\mathsf{F})$. Montrer que, pour toute formule F, les formules F et $\mathtt{red}(\mathsf{F})$ sont équivalentes.

(*ii*) Construire un algorithme décidant la validité de F en considérant successivement chacune des variables qui apparaissent dans F, et, pour chacune d'elles, en substituant les deux valeurs possibles $\top$ et $\bot$ et en réduisant. Appliquer à $(\boldsymbol{X}_1\Rightarrow(\boldsymbol{X}_2\vee\boldsymbol{X}_3))\Rightarrow((\boldsymbol{X}_1\Rightarrow\boldsymbol{X}_2)\vee(\boldsymbol{X}_1\Rightarrow\boldsymbol{X}_3))$. Quel est l'intérêt de la méthode?

Exercice 4. (formule croissante) Pour V, V' deux affectations, on dit qu'on a $V \leqslant V'$ si, en posant $0 < 1$, on a $V(\boldsymbol{X}_\mathsf{i}) \leqslant V'(\boldsymbol{X}_\mathsf{i})$ pour toute variable $\boldsymbol{X}_\mathsf{i}$. On dit qu'une formule propositionnelle F est *croissante* si, quand V satisfait F et qu'on a $V \leqslant V'$, alors V' satisfait F. Donner un exemple de formule F telle que ni F, ni $\neg\mathsf{F}$ ne soit croissante. Montrer que F est croissante si et seulement si elle est équivalente à une formule écrite sans $\neg$ ni $\Rightarrow$.

Exercice 5. (forme clausale) On dit qu'une formule propositionnelle F est une *clause* si F est de la forme $\mathsf{L}_1\vee\ldots\vee\mathsf{L}_\mathsf{n}$, où chaque formule L_k est une variable ou la négation d'une variable. Construire un algorithme qui associe à toute formule F une conjonction de clauses F' équivalente à F. [Indication: Remplacer les équivalences et les implications par des conjonctions, disjonctions, et négations, faire migrer les négations vers les variables, et les conjonctions vers la racine de l'arbre.]

Exercice 6. (preuves par résolution) (*i*) Pour $\mathsf{C}_1, \mathsf{C}_2, \mathsf{C}$ clauses propositionnelles (exercice 5), on dit que C est déduite de C_1 et C_2 par *résolution* en $\boldsymbol{X}_\mathsf{i}$ si $\boldsymbol{X}_\mathsf{i}$ apparaît dans C_1, $\neg\boldsymbol{X}_\mathsf{i}$ apparaît dans C_2, ou *vice versa*, et, en appelant C' le mot obtenu à partir de $\mathsf{C}_1\vee\mathsf{C}_2$ en déplaçant les parenthèses, en supprimant les occurrences de $\boldsymbol{X}_\mathsf{i}$ venant de C_1 et les occurrences de $\neg\boldsymbol{X}_\mathsf{i}$ venant de C_2 (ou *vice versa*), et en supprimant les répétitions, ou bien C' est non vide et on a $\mathsf{C} = \mathsf{C}'$, ou bien C' est vide, et on a $\mathsf{C} = \bot$. Soient $\mathsf{C}_1 = \neg\boldsymbol{X}_1\vee\boldsymbol{X}_2\vee\boldsymbol{X}_3$, $\mathsf{C}_2 = \boldsymbol{X}_1\vee\neg\boldsymbol{X}_3\vee\boldsymbol{X}_4$. Ecrire toutes les clauses pouvant s'obtenir à partir des clauses précédentes par résolution.

(*ii*) Montrer que, si T est un ensemble de clauses, et si C est une clause pouvant s'obtenir par résolution à partir de T, alors T est satisfaisable si et seulement si $\mathsf{T}\cup\{\mathsf{C}\}$ l'est aussi.

En déduire que, s'il existe une preuve par résolution de $\bot$ à partir de T, alors T n'est pas satisfaisable (cohérence de la résolution).

(*iii*) Montrer la réciproque (complétude de la résolution). [Indication : Notons $i(\mathsf{T})$ l'indice maximal d'une variable apparaissant dans T. On montrera par récurrence sur $i(\mathsf{T})$ que, si T est non satisfaisable, alors il existe une preuve par résolution de $\bot$ à partir de T. Pour cela, on construira T' non satisfaisable à partir de T vérifiant $i(\mathsf{T}') < i(\mathsf{T})$ en retirant toutes les clauses de T où $\boldsymbol{X}_{i(\mathsf{T})}$ figure mais en ajoutant toutes les clauses qui s'en déduisent par résolution en $\boldsymbol{X}_{i(\mathsf{T})}$. La méthode de résolution a un grand intérêt algorithmique.]

Exercice 7. (colorabilité) On dit qu'un graphe orienté (V, E) est k-colorable s'il existe une application f de V dans $\{1, \ldots, k\}$ telle que $(x, y) \in E$ entraîne $f(x) \neq f(y)$. Montrer qu'un graphe orienté (V, E) est k-colorable si et seulement si tout sous-graphe fini de (V, E) l'est. [Exprimer la colorabilité à l'aide de formules propositionnelles, et utiliser le théorème de compacité.]