Informatique *Cours*
Logique Ipesup

Objectifs

À l'issue de cette leçon, l'étudiant devra être capable de :
- comprendre la syntaxe et la sémantique de la logique propositionnelle, ainsi que les notions de valuation, modèle et satisfiabilité ;
- manipuler et transformer des formules logiques à l'aide des équivalences sémantiques et des formes normales (NNF, CNF, DNF).

L'objet de la *logique* est la formalisation du *discours* et du *raisonnement*. On attribue à Aristote l'une des premières tentatives de formalisation du raisonnement, à l'aide de ce qu'on appelle la logique des syllogismes. La logique définit formellement à la fois le langage que l'on utilise (aspects *syntaxiques*), et sa signification ou *interprétation* (aspects *sémantiques*). La logique manipule des objets appelés *formules*, qui sont des objets structurés, construits à partir de *propositions élémentaires* articulées par des *connecteurs logiques* et des *quantificateurs*.

En informatique, la logique est le cadre naturel pour aborder les *problèmes de décision*. Ces derniers tentent de répondre en termes algorithmiques à la question de l'existence d'une solution à un problème. De fait, la logique permet d'une part la formalisation d'un tel problème par des formules logiques, et d'autre part la mise en œuvre d'algorithmes permettant l'analyse et la résolution de ces formules.

La *logique propositionnelle* combine des faits élémentaires qui ne peuvent prendre que deux valeurs : *vrai* ou *faux*. Par exemple, chacune des phrases suivantes affirme un fait, et ce fait peut être vrai, ou être faux :

p_1 : Il pleut.
p_2 : Je prends mon parapluie.

En combinant de telles assertions, on produit de nouveaux énoncés dont le caractère de vérité est discuté. Des *connecteurs logiques* permettent la construction de ces énoncés, à l'image des conjonctions de coordination qui lient ou modifient le sens des phrases. Par exemple :

φ_1 : Il pleut et je prends mon parapluie.
φ_2 : Il pleut ou je prends mon parapluie.
φ_3 : Puisqu'il pleut, je ne prends pas mon parapluie.
φ_4 : Si je prends mon parapluie alors il pleut.

Toutes ces phrases sont correctes d'un point de vue *syntaxique* même si leur *sémantique* est parfois étrange. Les deux phrases φ_1 et φ_2 sont construites à partir des phrases élémentaires p_1 et p_2, liées par les mots *et* et *ou*. On parle de phrase *conjonctive* pour φ_1 et de phrase *disjonctive* pour φ_2. Les phrases φ_3 et φ_4 expriment l'idée d'une *implication*. La phrase φ_3 contient également une *négation* de la phrase p_2, c'est-à-dire une phrase de sens contraire.

En logique, des notations permettent une écriture plus compacte de ces combinaisons :

$$\varphi_1 = (p_1 \wedge p_2)$$
$$\varphi_2 = (p_1 \vee p_2)$$
$$\varphi_3 = (p_1 \to (\neg p_2))$$
$$\varphi_4 = (p_2 \to p_1)$$

Les symboles $\wedge$, $\vee$ et $\to$ sont des *connecteurs binaires*, qui permettent la construction d'un nouvel énoncé à partir de deux énoncés.
Le premier réalise une *conjonction* : il relie deux énoncés qui doivent être tous deux vrais. Il est nommé *et*.
Le deuxième réalise une *disjonction* : il relie deux énoncés dont l'un au moins doit être vrai. Il est nommé *ou*.
Le troisième réalise une *implication* : il relie deux énoncés en exprimant que, dès lors que le premier est vrai, le deuxième doit l'être aussi.
Le connecteur unaire $\neg$ s'applique à un unique énoncé pour en exprimer le contraire. Il s'agit de la *négation*. Ainsi, ces notations rendent plus synthétique l'écriture des phrases.

Mais ce n'est pas là leur seul intérêt. Les connecteurs logiques explicitent la manière dont les différents éléments d'une phrase sont articulés, de sorte à en retirer toutes les ambiguïtés propres au langage courant, pour permettre ensuite un raisonnement rigoureux.

Par l'abstraction qu'ils apportent, les connecteurs donnent aussi des énoncés pouvant présenter un intérêt plus large que celui du contexte qui a mené à leur écriture. Ainsi, quelle que soit l'assertion désignée par la lettre p, l'expression $\neg(p \wedge (\neg p))$ exprime plus largement qu'une information ne peut être à la fois vraie et fausse. La logique propositionnelle va donc s'attacher à analyser des structures logiques indépendamment de leur problème d'origine.

Mais la logique propositionnelle est incapable d'exprimer l'existence d'un objet ayant une propriété donnée ou encore le fait que plusieurs objets partagent une même propriété. Une proposition telle que p_1 ou p_2 a une structure interne, que l'on peut également intégrer à la logique. La *logique du premier ordre*, également appelée *logique des prédicats*, permet cela. Dans les énoncés suivants :

Le ciel est bleu.

L'encre est bleue.

le sujet est un argument qualifié par son attribut *est bleu*. En adoptant une notation synthétique P pour exprimer l'idée d'*être bleu*, on peut ré-écrire les énoncés sous la forme P(encre) et P(ciel). P est appelé un *prédicat unaire*. Les prédicats nous donnent donc une granularité plus fine dans la structuration du discours, et vont permettre ainsi l'écriture d'énoncés plus généraux. Et il sera toujours possible de relier ces énoncés entre eux par les connecteurs déjà connus.

L'inégalité $(x < 10)$, où x est un nombre entier, peut s'écrire à l'aide d'un *prédicat binaire* Q qui exprime l'idée *être strictement inférieur à* : $Q(x, 10)$. On peut ensuite construire des énoncés complexes, en combinant de tels prédicats à l'aide des connecteurs logiques. Ainsi on peut traduire la propriété selon laquelle un entier x vérifie $2 \leq x < 6$ par une expression de la forme $\neg Q(x, 2) \wedge Q(x, 6)$. En ce sens, la logique du premier ordre, étend donc le champ de la logique des propositions en enrichissant son discours.

Elle va même plus loin, en introduisant deux *quantificateurs* universel $\forall$ et existentiel $\exists$, qui de manières différentes donnent leur sens à des objets indéterminés comme le « x » de l'énoncé

précédent. Par exemple, comment exprimer sous forme logique la déduction suivante ?

Tous les hommes sont mortels.
Socrate est un homme.
Donc Socrate est mortel.

La logique des propositions en est incapable, du fait de la présence dans ces phrases de prédicats. La phrase *Tous les hommes sont mortels* affecte l'attribut *mortels* au sujet *les hommes*, par l'intermédiaire du verbe *être*. Ce sont alors les groupes situés en position de sujet et d'attribut qui sont les nouveaux atomes de nos phrases, et doivent être combinés pour former des propositions élémentaires. On dit que la phrase a une *structure prédicative*. En outre, certaines de ces expressions possèdent un caractère *quantitatif* exprimé par *Tous les ...* ou encore par *Il existe un ...*. Les quantificateurs vont permettre l'écriture de ces énoncés qui dépendent d'éléments variables. Ainsi, en désignant par M le prédicat *être mortel* et par H le prédicat *être un homme*, ces phrases peuvent se représenter par $M(\text{Tous les hommes})$, par $H(\text{Socrate})$ et par $M(\text{Socrate})$, de sorte que le raisonnement s'exprime par une formule logique :

$$(M(\text{Tous les hommes}) \wedge H(\text{Socrate})) \to M(\text{Socrate})$$

On peut aller un peu plus loin dans la formalisation en décomposant le prédicat $M(\text{Tous les hommes})$, pour qu'il ne s'applique plus qu'à un individu et non toute une population. Alors $M(\text{Tous les hommes})$ se ré-écrit $\forall \mathit{homme}.\, M(\mathit{homme})$, qu'on lit « pour tout homme, cet homme est mortel », et qu'il faut comprendre comme « tout homme que l'on puisse considérer, est mortel ». On termine la construction en reliant $\forall \mathit{homme}$ au prédicat H qui justement caractérise les hommes, et on obtient :

$$(\forall x.\, (H(x) \to M(x)) \wedge H(s)) \to M(s)$$

où x et s sont des *variables*, au sens général du terme, et où s peut être instancié, par exemple, par Socrate.

On introduit quelques aspects de la logique en lien avec quelques problèmes informatiques. On develope la *syntaxe* des logiques propositionnelle et du premier ordre, puis la *sémantique* de la logique propositionnelle qui mène naturellement au problème « SAT » de la résolution d'une formule. Enfin, la *déduction naturelle* formalise la notion de *démonstration*.

1 Logique propositionnelle

1.1 Variable et formule propositionnelles

La *logique propositionnelle* étudie les propriétés d'énoncés complexes construits à partir d'énoncés élémentaires qui ne peuvent être que *vrais* ou *faux*. Son objectif est de donner un *sens*, appelé *valeur de vérité*, à ces énoncés complexes sous réserve qu'ils soient *bien écrits*. Il convient donc, dans un premier temps, de préciser les règles qui régissent la construction d'énoncés *syntaxiquement corrects* pour, dans un second temps, étudier leur *sémantique*.

Variable propositionnelle

Une *variable propositionnelle* (ou *proposition atomique*) est une assertion qui ne peut prendre que deux états possibles appelés *valeurs de vérité*.

On note $\mathcal{V}$ l'ensemble des variables propositionnelles, désignées par une lettre minuscule : x, y, z.

Deux symboles $\top$ et $\bot$, appelés *constantes logiques*, désignent respectivement une proposition *toujours vraie* et une proposition *toujours fausse*. Ils forment, avec les variables propositionnelles, les briques élémentaires des énoncés logiques. Les formules logiques sont construites inductivement, en prenant comme objets de base les variables propositionnelles et les deux constantes $\top$ et $\bot$, et en les combinant par des *connecteurs logiques*.

Formule logique

Soit la signature contenant :
- les constantes $\top$, $\bot$ et l'ensemble des variables propositionnelles $\mathcal{V}$;
- un constructeur unaire `not` représentant la négation ;
- trois constructeurs binaires `and`, `or`, `imp` représentant le *et*, le *ou* et l'*implication*.

Une *formule logique* est un *terme* sur cette signature.

Si x et y sont deux variables propositionnelles, les expressions suivantes sont des formules logiques :

$$\texttt{and}(x, y) \qquad \texttt{or}(x, \bot) \qquad \texttt{and}(\texttt{not}(x), y) \qquad \texttt{imp}(x, \texttt{or}(\top, y))$$

1.1.1 OCaml

Une telle définition mène très naturellement à la définition d'un type de données récursif dans un langage comme OCaml. Il suffit pour cela d'introduire un constructeur pour chaque forme possible d'énoncé logique. Un fragment pourrait en être :

```
type fmla =
  | True
  | ...
  | And of fmla * fmla
  | Or  of fmla * fmla
  | Imp of fmla * fmla
```

Pour limiter les redondances dans les définitions et le code, nous allons légèrement modifier ce schéma général en regroupant tous les connecteurs binaires sous une même construction. Le programme suivant définit ainsi deux types : un type `binop` qui est l'énumération des trois connecteurs binaires, et un type `fmla` pour les formules elles-mêmes.

```
type binop = And | Or | Imp

type fmla =
  | True
  | False
  | Var of int (* dans 1..n *)
  | Not of fmla
  | Bin of binop * fmla * fmla
```

La formule logique

$$\varphi = \texttt{and}(\texttt{or}(\texttt{imp}(x, y), \texttt{and}(\texttt{not}(x), y)), \texttt{or}(x, \texttt{not}(y)))$$

peut ainsi se définir comme suit :

```
let x = Var 1 and y = Var 2
let f0 = Bin (Imp, x, y)
let f1 = Bin (And, Not x, y)
let f2 = Bin (Or, x, Not y)
let phi = Bin (And, Bin (Or, f0, f1), f2)
```

Pour manipuler les formules logiques, le constructeur `Var` est suivi d'un entier naturel non nul. CCe choix n'est pas arbitraire : il est motivé par l'usage des *SAT-solvers*, programmes destinés à résoudre automatiquement le problème de satisfiabilité propositionnelle (SAT). Ces solveurs manipulent des formules sous forme normale conjonctive (CNF) et utilisent généralement le format standard *DIMACS*, dans lequel chaque variable propositionnelle est représentée par un entier naturel non nul, et sa négation par l'entier opposé. Cette représentation permet une manipulation algorithmique efficace des formules logiques. En adoptant cette convention, pour nos besoins, il peut être utile de déterminer l'ensemble des variables. Le programme suivant renvoie le plus grand entier associé à une variable propositionnelle d'une formule.

```
let varmax f =
  let rec varmax m = function
    | True | False -> m
    | Var i -> max i m
    | Not f -> varmax m f
    | Bin (_, f1, f2) -> varmax (varmax m f1) f2
  in
  varmax 0 f
```

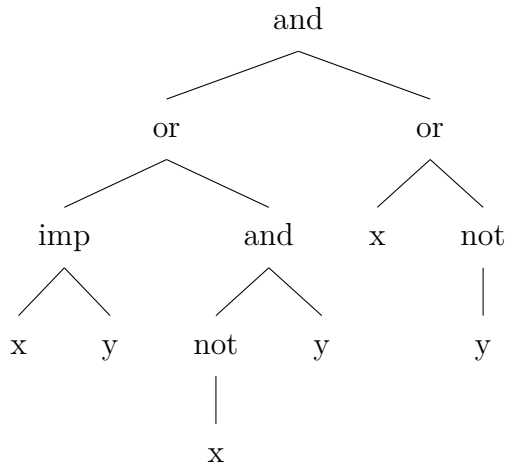
Arbre de syntaxe abstraite. Le type OCaml `fmla` est ce qu'on appelle un *arbre de syntaxe abstraite* d'une formule logique. Il s'agit d'un arbre fini non vide dont les feuilles sont des variables propositionnelles et les nœuds de l'arbre portent les connecteurs logiques.

- Pour toute formule φ, l'arbre de syntaxe abstraite associé à $\texttt{not}(\varphi)$ a une racine étiquetée par `not` et un unique enfant qui est l'arbre de syntaxe abstraite associé à φ.
- Pour toutes formules φ et ψ et tout connecteur binaire $c \in \{\texttt{and}, \texttt{or}, \texttt{imp}\}$, l'arbre de syntaxe abstraite associé à $c(\varphi, \psi)$ a une racine étiquetée par c, un sous-arbre gauche associé à φ et un sous-arbre droit associé à ψ.

Considérons la formule logique φ définie plus haut par :

$$\texttt{and}(\texttt{or}(\texttt{imp}(x, y), \texttt{and}(\texttt{not}(x), y)), \texttt{or}(x, \texttt{not}(y)))$$

Son arbre de syntaxe abstraite est le suivant.



Notons que chaque sous-arbre définit une formule appelée *sous-formule* de la formule initiale. Les sous-formules associées sont :

$$\texttt{or}(\texttt{imp}(x,y),\texttt{and}(\texttt{not}(x),y)),\quad \texttt{or}(x,\texttt{not}(y)),\quad \texttt{not}(y)$$

À chaque formule logique φ, on peut associer deux entiers, sa *taille* $|\varphi|$ et sa *hauteur* $h(\varphi)$, qui permettent de mener des raisonnements par induction et de prouver par récurrence certaines propriétés de la formule.

Taille d'une formule

La taille d'une formule φ, notée $|\varphi|$, est définie inductivement par :

$$\begin{aligned}
|\top| &= 0 \\
|\bot| &= 0 \\
|x| &= 0 \qquad (x \text{ variable propositionnelle}) \\
|\texttt{not}(\varphi)| &= 1 + |\varphi| \\
|c(\varphi,\psi)| &= 1 + |\varphi| + |\psi|
\end{aligned}$$

où c est un connecteur binaire.

On peut remarquer que la taille d'une formule est aussi le nombre de connecteurs qu'elle contient.

Hauteur d'une formule

La hauteur d'une formule φ, notée $h(\varphi)$, est définie inductivement par :

$$\begin{aligned}
h(\top) &= 0 \\
h(\bot) &= 0 \\
h(x) &= 0 \qquad (x \text{ variable propositionnelle}) \\
h(\texttt{not}(\varphi)) &= 1 + h(\varphi) \qquad (\varphi \text{ formule logique}) \\
h(c(\varphi,\psi)) &= 1 + \max(h(\varphi), h(\psi))
\end{aligned}$$

où φ et ψ sont des formules logiques et c un connecteur binaire.

Formule linéaire. Il existe une écriture *linéaire* des formules logiques à l'aide des variables propositionnelles, des connecteurs et de parenthèses, qui est un peu plus légère à manipuler que la notation stricte à base de constructeurs. Dans cette représentation, les connecteurs sont représentés par les symboles :

— $\neg$ pour le constructeur `not` ;
— $\wedge$, $\vee$, $\rightarrow$ pour les constructeurs `and`, `or`, `imp`.

Alors que les constructeurs binaires sont utilisés comme des *opérateurs préfixes*, leurs symboles équivalents précédents sont utilisés de manière *infixe*.
Ainsi, la formule φ définie plus haut par :

$$\texttt{and}(\texttt{or}(\texttt{imp}(x, y), \texttt{and}(\texttt{not}(x), y)), \texttt{or}(x, \texttt{not}(y)))$$

peut être représentée par la formule linéaire suivante :

$$(((x \rightarrow y) \vee (\neg x \wedge y)) \wedge (x \vee \neg y))$$

Une définition alternative mais équivalente d'une *formule logique* serait alors la suivante.

— $\top$ et $\bot$ sont des formules logiques.
— Toute variable propositionnelle est une formule logique.
— Si φ est une formule logique alors $\neg\varphi$ est une formule logique.
— Si φ et ψ sont des formules logiques alors pour tout connecteur binaire $\diamond$, $(\varphi \diamond \psi)$ est une formule logique.

La première forme introduite pour les formules pouvait déjà être qualifiée de linéaire dans le sens où elle était écrite sur une ligne ! Mais elle n'est en réalité rien d'autre qu'un arbre. Le qualificatif est donc préféré pour désigner une formule mise sous la forme précédente qui n'est pas naturellement un arbre.

Si φ et ψ sont deux formules logiques, on a les notations suivantes.

Terme	Notation usuelle
$\texttt{not}(\varphi)$	$\neg\varphi$
$\texttt{and}(\varphi, \psi)$	$(\varphi \wedge \psi)$
$\texttt{or}(\varphi, \psi)$	$(\varphi \vee \psi)$
$\texttt{imp}(\varphi, \psi)$	$(\varphi \rightarrow \psi)$

Sous cette forme, les parenthèses jouent un rôle essentiel pour fixer les priorités des opérations. Si certaines peuvent sembler superflues, pour des expressions plus complexes, elles sont indispensables pour éviter toute ambiguïté. Par exemple, comment lire l'expression $x \wedge y \vee z$? Les expressions $(x \wedge y) \vee z$ et $x \wedge (y \vee z)$ sont non ambiguës. Et en toute rigueur, pour coller parfaitement à la définition précédente d'une formule linéaire, il conviendrait d'ajouter un couple de parenthèses pour l'ensemble de l'expression.
Ce qui mènerait à l'écriture de *formules strictes*.

$$((x \wedge y) \vee z) \qquad (x \wedge (y \vee z))$$

En pratique, ces parenthèses externes peuvent être omises sans que cela ne nuise à la syntaxe de la formule. Ainsi, les expressions $(x \leftrightarrow (\neg z \vee y))$ et $x \leftrightarrow (\neg z \vee y)$ sont syntaxiquement correctes. Elles comportent toutes les parenthèses indispensables. Notons que le connecteur unaire $\neg$ ne requiert pas nécessairement l'usage de parenthèses.

En revanche, les expressions $x \vee y$, $(x \vee y)$, $(x \vee y(etx \vee y))$, syntaxiquement incorrectes, ne sont pas des formules logiques.

Dans cette représentation, une *sous-formule* est une suite de symboles qui est encore une formule, c'est-à-dire une formule linéaire syntaxiquement correcte.
Par exemple, $(x \to y)$, $(\neg x \wedge y)$, $(x \vee \neg y)$, $((x \to y) \vee (\neg x \wedge y))$ sont des sous-formules de la formule $(((x \to y) \vee (\neg x \wedge y)) \wedge (x \vee \neg y))$.

Toute formule logique se décompose de manière unique en sous-formules. L'unicité de cette décomposition implique qu'on peut identifier une formule et son arbre de syntaxe abstraite, et une sous-formule et un sous-arbre de syntaxe abstraite de l'arbre de syntaxe abstraite. Ce résultat constitue le *théorème de lecture unique des formules*.

Theoreme de lecture unique des formules strictes

Toute écriture d'une formule stricte φ a exactement l'une des formes suivantes.
- φ est une variable propositionnelle.
- $\varphi = \neg\psi$ où ψ est une formule.
- $\varphi = (\psi_1 \diamond \psi_2)$ où ψ_1 et ψ_2 sont des formules, $\diamond$ est un connecteur binaire.

Dans ces deux derniers cas, il y a unicité des formules ψ, ψ_1 et ψ_2.

Règles de priorités Pour alléger l'écriture des formules logiques, certaines parenthèses, voire toutes, peuvent être supprimées sans générer d'ambiguïté de lecture de la formule si certaines *règles de priorités* sont adoptées, comparables aux règles de priorités usuelles de l'arithmétique.

- Le connecteur $\neg$ est prioritaire sur tous les autres connecteurs.
- Puis, dans l'ordre des priorités décroissantes, on a
  - $\wedge$,
  - $\vee$;
  - $\to$

Exemple

Par exemple, la formule $((x \wedge y) \vee z)$ peut s'écrire $x \wedge y \vee z$, sans ambiguïté de lecture. Toutefois, la présence des parenthèses internes donne plus de lisibilité à la formule. Un bon compromis est donc $(x \wedge y) \vee z$.

- Quand plusieurs mêmes connecteurs se suivent, celui situé le plus à gauche est prioritaire. On parle d'*associativité à gauche*. Cette règle admet une exception pour le connecteur d'implication $\to$, pour lequel l'associativité est à droite.

Exemple

La formule $x \to y \to z$ se lit $(x \to (y \to z))$, alors que la formule $x \wedge y \wedge z$ se lit $((x \wedge y) \wedge z)$.

1.2 Sémantique

En linguistique, la *syntaxe* désigne le *signifiant* d'un énoncé. La *sémantique* désigne son *signifié*. Il existe entre la syntaxe et la sémantique le même rapport qu'entre la forme et le fond. La syntaxe est le support de la sémantique.

En logique propositionnelle, la *sémantique* s'attache à définir la *valeur de vérité* d'une formule syntaxiquement correcte, à savoir son caractère *vrai* ou *faux*. Pour ce faire, il convient :

— d'attribuer une valeur de vérité à chaque variable propositionnelle ;
— de définir les règles d'interprétation d'un connecteur ;
— de déterminer la valeur de vérité de la formule.

1.3 Valuation

On appelle *ensemble des booléens* $\mathbb{B}$ l'ensemble $\{F, V\}$. D'autres notations sont possibles pour désigner les valeurs de cet ensemble, comme 0 pour faux et 1 pour vrai, ou encore `false` et `true`.

Valuation

On appelle *valuation* (ou *environnement*, ou *distribution de vérité*, ou *contexte*) toute fonction $v : \mathcal{V} \to \mathbb{B}$. Une valuation est donc un choix de valeurs de vérité attribuées à chacune des variables propositionnelles d'une formule.

Choisir une valuation v associée à un triplet de variables (x, y, z), c'est par exemple imposer :

$$v(x) = F \qquad v(y) = V \qquad v(z) = F$$

On voit que, pour un tel triplet, 2^3 valuations peuvent être définies. De manière générale, si une formule comporte n variables propositionnelles, il existe 2^n choix de valuations possibles. Comme nous le verrons par la suite, ce résultat revêt une grande importance.

La connaissance d'une valuation des variables propositionnelles d'une formule permet de déterminer la valeur de vérité de cette dernière. Comme elle est construite à l'aide de connecteurs logiques, il convient tout d'abord de préciser les règles d'interprétation de ces derniers.

Fonction booléenne

On appelle *fonction booléenne à n arguments* toute fonction de $\mathbb{B}^n$ dans $\mathbb{B}$. Ainsi, à chaque connecteur peut être associée une fonction booléenne qui exprime la valeur de vérité d'une formule connaissant celle de ses sous-formules.

— La fonction $f_\neg : \mathbb{B} \to \mathbb{B}$ est définie par :

$$f_\neg(F) = V \qquad f_\neg(V) = F$$

— La fonction $f_\wedge : \mathbb{B}^2 \to \mathbb{B}$ est définie par :

$$f_\wedge(x, y) = \begin{cases} V & \text{si et seulement si } x = V \text{ et } y = V \\ F & \text{sinon} \end{cases}$$

— La fonction $f_\vee : \mathbb{B}^2 \to \mathbb{B}$ est définie par :

$$f_\vee(x, y) = \begin{cases} F & \text{si et seulement si } x = F \text{ et } y = F \\ V & \text{sinon} \end{cases}$$

— La fonction $f_{\rightarrow} : \mathbb{B}^2 \rightarrow \mathbb{B}$ est définie par :

$$f_{\rightarrow}(x, y) = \begin{cases} F & \text{si et seulement si } x = V \text{ et } y = F \\ V & \text{sinon} \end{cases}$$

— La fonction $f_{\leftrightarrow} : \mathbb{B}^2 \rightarrow \mathbb{B}$ est définie par :

$$f_{\leftrightarrow}(x, y) = \begin{cases} F & \text{si et seulement si } x \neq y \\ V & \text{sinon} \end{cases}$$

Ces résultats peuvent être exprimés sous la forme de tableaux appelés *tables de vérité*. Chaque ligne du tableau correspond à l'une des valuations, et est constituée : de la valeur donnée par la valuation à chaque variable propositionnelle argument d'une fonction booléenne, et la valeur de vérité correspondant au résultat de la fonction. Voici les tables de vérité associées aux fonctions d'interprétation des connecteurs logiques.

p	$f_{\neg}(p)$
F	V
V	F

x	y	$f_{\vee}(x, y)$	$f_{\wedge}(x, y)$	$f_{\rightarrow}(x, y)$	$f_{\leftrightarrow}(x, y)$
F	F	F	F	V	V
F	V	V	F	V	F
V	F	V	F	F	F
V	V	V	V	V	V

1.4 Valeur d'une formule

La valeur de vérité d'une formule peut être définie inductivement à partir de celle de ses variables propositionnelles et des fonctions booléennes précédentes.

Valeur d'une formule

Soit $\mathcal{V}$ un ensemble de variables propositionnelles et $\mathcal{F}$ l'ensemble des formules logiques qu'il est possible de construire sur $\mathcal{V}$. Pour toute valuation v, la *fonction d'évaluation d'une formule* $[\![.]\!]_v : \mathcal{F} \rightarrow \mathbb{B}$ se définit par induction structurelle :

$$\begin{aligned} [\![x]\!]_v &= v(x) \\ [\![\neg\varphi]\!]_v &= f_{\neg}([\![\varphi]\!]_v) \qquad (\varphi \text{ formule logique}) \\ [\![\varphi \diamond \psi]\!]_v &= f_{\diamond}([\![\varphi]\!]_v, [\![\psi]\!]_v) \end{aligned}$$

où φ et ψ sont des formules logiques et $\diamond$ un connecteur binaire.

Notation abrégée de la fonction d'évaluation Pour toute formule φ et toute valuation v, on s'autorise donc à écrire $v(\varphi)$ pour désigner $[\![\varphi]\!]_v$.

Étant donnée une valuation v sur les variables propositionnelles d'une formule φ, la valeur de $v(\varphi)$ ne dépend que de la valeur de v en les variables propositionnelles ayant occurrence

dans φ. Si les variables propositionnelles intervenant dans φ sont $x_1, x_2, \ldots, x_n$, il suffit de considérer les valuations restreintes à $\{x_1, x_2, \ldots, x_n\}$ pour connaître toutes celles de φ.

Exemple

Reprenons la formule φ définie plus haut.

$$(((x \to y) \vee (\neg x \wedge y)) \wedge (x \vee \neg y))$$

Adoptons une valuation v définie par $v(x) = F$, $v(y) = F$. Alors :

$$\begin{aligned}
v(\varphi) &= v((((x \to y) \vee (\neg x \wedge y)) \wedge (x \vee \neg y))) \\
&= f_\wedge\Big(v(((x \to y) \vee (\neg x \wedge y))), v(x \vee \neg y)\Big) \\
&= f_\wedge\Big(f_\vee(v(x \to y), v(\neg x \wedge y)), f_\vee(v(x), v(\neg y))\Big) \\
&= f_\wedge\Big(f_\vee(f_\to(v(x), v(y)), f_\wedge(v(\neg x), v(y))), f_\vee(v(x), f_\neg(v(y)))\Big) \\
&= f_\wedge\Big(f_\vee(f_\to(F, F), f_\wedge(f_\neg(F), F)), f_\vee(F, f_\neg(F))\Big) \\
&= f_\wedge\Big(f_\vee(V, f_\wedge(V, F)), f_\vee(F, V)\Big) \\
&= f_\wedge\Big(f_\vee(V, F), V\Big) \\
&= f_\wedge(V, V) \\
&= V
\end{aligned}$$

Comme nous l'avons déjà évoqué plus haut, pour toute formule comportant n variables propositionnelles, il existe exactement 2^n valuations.

Toutes ces valuations peuvent être présentées dans la *table de vérité* de φ comportant 2^n lignes.

Sur chacune de ses lignes sont portées les valuations attribuées à chaque variable propositionnelle, puis celle de chaque sous-formule, et enfin celle de la formule en dernière colonne.

Pour alléger les écritures, on omet généralement la notation ().

Exemple

Toujours avec la même formule φ,

$$(((x \to y) \vee (\neg x \wedge y)) \wedge (x \vee \neg y))$$

la table de vérité est la suivante.

x	y	$x \to y$	$\neg x \wedge y$	$(x \to y) \vee (\neg x \wedge y)$	$x \vee \neg y$	φ
F	F	V	F	V	V	V
F	V	V	V	V	F	F
V	F	F	F	F	V	F
V	V	V	F	V	V	V

En OCaml, l'évaluation d'une formule peut se faire à l'aide d'une fonction `eval` à deux arguments. Le premier argument est un tableau de booléens qui associe une valeur de vérité à chaque variable propositionnelle. La case 0 du tableau est inutilisée de sorte que les variables sont identifiées par des entiers naturels non nuls commençant à 1. Le second argument de la fonction est une formule dont le type est celui du programme 10.1.

```
let rec eval v f = match f with
| True -> true
| False -> false
| Var i -> assert (1 <= i && i < Array.length v); v.(i)
| Not f -> not (eval v f)
| Bin (And, f1, f2) -> eval v f1 && eval v f2
| Bin (Or, f1, f2) -> eval v f1 || eval v f2
| Bin (Imp, f1, f2) -> not (eval v f1) || eval v f2
```

1.5 Modèle d'une formule.

Parmi les valuations d'une formule logique φ, on distingue celles pour lesquelles la formule est vraie et celles pour lesquelles elle est fausse.
Une valuation qui rend vraie une formule est appelée un *modèle* pour cette formule. On peut alors définir un ensemble de toutes les valuations qui rendent une formule vraie. L'ensemble des modèles d'une formule porte autant d'informations que sa table de vérité.

Modèle

Étant donnée une formule φ, un *modèle* de cette formule est une valuation v qui rend vraie φ. On note $\mathcal{M}_v$ un tel modèle.

La notation $v \models \varphi$ est parfois adoptée pour signifier que v est un modèle de φ.
On la lit également : φ est satisfaite par la valuation v.

Satisfiabilité d'une formule

Une formule φ est dite *satisfiable* s'il existe une valuation qui la rend vraie, c'est-à-dire s'il existe une valuation v telle que $v \models \varphi$.

La notion de satisfiabilité est fondamentale en logique. Savoir si une formule est satisfiable constitue le cœur du problème SAT. Ainsi, l'ensemble des modèles d'une formule φ n'est autre que l'ensemble des valuations qui satisfont la formule. On peut noter cet ensemble $\mathrm{Mod}(\varphi)$.

Si $\mathcal{V}$ est l'ensemble des variables propositionnelles sur lequel est défini φ, on peut écrire :

$$\mathrm{Mod}(\varphi) = \{\mathcal{M}_v \mid v \in \mathbb{B}^{\mathcal{V}}\}$$

ou encore :

$$\mathrm{Mod}(\varphi) = \{v \in \mathbb{B}^{\mathcal{V}} \mid v \models \varphi\}$$

Tautologie et antilogie

Une formule logique φ satisfaite pour toute valuation de ses variables propositionnelles est appelée une *tautologie*. On dit également que la formule est *valide*. On note $\models \varphi$. Si aucune valuation ne satisfait une formule, cette dernière est appelée *antilogie*, et on la dit aussi *contradictoire*.

Exemple

Pour toute variable propositionnelle x, la formule $x \vee \neg x$ est une tautologie. En revanche, la formule $x \wedge \neg x$ est une antilogie.

x	$x \vee \neg x$	$x \wedge \neg x$
F	V	F
V	V	F

De cette définition, il découle immédiatement que φ est une tautologie si et seulement si $\neg\varphi$ est une antilogie. En effet, si on note Val l'ensemble de toutes les valuations possibles sur l'ensemble des variables propositionnelles $\mathcal{V}$, φ est une tautologie si et seulement si $\text{Mod}(\varphi) = \text{Val}$. Cette égalité équivaut à $\text{Val} \setminus \text{Mod}(\varphi) = \varnothing$.

En remarquant que $\text{Mod}(\neg\varphi) = \text{Val} \setminus \text{Mod}(\varphi)$, on a finalement $\text{Mod}(\neg\varphi) = \varnothing$, c'est-à-dire que $\neg\varphi$ est insatisfiable (antilogie).

1.6 Conséquence logique

La notion de *conséquence* peut prendre plusieurs sens, exprimés par les symboles $\rightarrow$, $\vDash$ et $\Rightarrow$. Pour clarifier ces notations, considérons la phrase : « S'il pleut, je prends mon parapluie. ». Notons x la proposition *il pleut* et y la proposition *je prends mon parapluie*, et posons $\varphi = (x \rightarrow y)$.

Dans le langage usuel, cette formule est interprétée comme affirmant que, s'il est établi qu'il pleut, alors il est vrai que je prends mon parapluie. Cependant, cette interprétation dépasse ce que signifie réellement la formule φ. Cette dernière relève seulement de ce qu'on pourrait appeler le *langage-objet*, langage dénué de toute sémantique. Dit autrement, on ne sait rien des valeurs de vérité de x et de y. La formule φ est seulement une expression qui définit une relation de *conséquence matérielle*.

La *conséquence sémantique* est à rapprocher de l'interprétation précédente de la phrase. On la note $x \vDash y$, relation qui se peut traduire par : *S'il est vrai qu'il pleuve alors il s'ensuit qu'il est vrai que je prenne mon parapluie*. Cette relation relève du *métalangage* et non du langage-objet.

Le symbole $\Rightarrow$ s'inscrit également dans ce cadre méta-linguistique : il indique que, si la formule φ est vraie, on peut écrire $x \Rightarrow y$.

De manière générale, lorsqu'une formule logique ψ est vraie chaque fois qu'une autre formule φ l'est, on dit que ψ est une *conséquence sémantique* de φ. Cette notion s'étend naturellement à un ensemble de formules Γ : si toute valuation qui satisfait toutes les formules de Γ satisfait également ψ, alors ψ est conséquence sémantique de Γ. Dans ces deux cas, on utilise la notation $\vDash$.

Conséquence sémantique

Une formule ψ est *conséquence sémantique* d'une formule φ si pour toute valuation v telle que $v(\varphi) = V$, alors $v(\psi) = V$. On note : $\varphi \vDash \psi$.
Une formule ψ est *conséquence sémantique* d'un ensemble de formules Γ quand, pour toute valuation v, si v est telle que toute formule $\varphi \in \Gamma$ vérifie $v(\varphi) = V$, alors $v(\psi) = V$. On note : $\Gamma \vDash \psi$.

Exemple

Illustrons la conséquence sémantique avec l'ensemble $\Gamma = \{x, y\}$ contenant deux formules réduites à des variables propositionnelles.
Il est clair que $\Gamma \vDash (x \wedge y)$. Toute valuation qui satisfait x et y satisfait leur conjonction.
On pourrait également écrire $\Gamma \vDash (x \vee y)$ même si la satisfiabilité de *toutes* les variables propositionnelles de Γ est une condition trop forte.
On a aussi $x \vDash (x \vee y)$ et $y \vDash (x \vee y)$.
Si à présent, on prend $\Gamma = \{(x \to y), (y \to z)\}$, on a $\Gamma \vDash (x \to z)$.
En effet, une valuation v qui satisfait $(x \to y)$ vérifie $v(x) = F$ ou $v(y) = V$.
En outre, une telle valuation satisfait $(y \to z)$ et donc vérifie $v(y) = F$ ou $v(z) = V$.
Deux choix sont possibles pour la valeur de vérité de y.
Si $v(y) = V$, alors nécessairement $v(z) = V$. Si $v(y) = F$ alors nécessairement $v(x) = F$.
Dans les deux cas, on obtient $v((x \to z)) = V$. Ce qui établit le résultat.

1.7 Équivalence sémantique

Quand deux formules logiques syntaxiquement différentes ont la même table de vérité, elles partagent la même sémantique. De fait, elles sont sémantiquement indiscernables bien que syntaxiquement discernables.

Équivalence sémantique

Deux formules φ et ψ sont dites *équivalentes* si pour toute distribution de vérité v, on a $v(\varphi) = v(\psi)$. On note alors $\varphi \equiv \psi$.

L'équivalence sémantique ne doit pas être confondue avec l'équivalence matérielle $\leftrightarrow$. Alors que $(\varphi \leftrightarrow \psi)$ est une formule, $\varphi \equiv \psi$ n'en est pas une. Cette dernière est seulement un jugement porté sur les formules φ et ψ qui exprime que d'un point de vue sémantique, elles sont indiscernables.

On peut néanmoins remarquer que deux formules satisfont $\varphi \equiv \psi$, si et seulement si $(\varphi \leftrightarrow \psi)$ est une tautologie, puisque pour toute valuation v telle que $v(\varphi) = v(\psi)$, on a $\varphi \equiv \psi$, et réciproquement par définition de l'équivalence sémantique. La table de vérité ci-dessous prouve que :

$$(\varphi \leftrightarrow \psi) \equiv (\varphi \to \psi) \wedge (\psi \to \varphi)$$

φ	ψ	$\varphi \to \psi$	$\psi \to \varphi$	$(\varphi \to \psi) \wedge (\psi \to \varphi)$	$\varphi \leftrightarrow \psi$
F	F	V	V	V	V
F	V	V	F	F	F
V	F	F	V	F	F
V	V	V	V	V	V

Équivalences sémantiques propositionnelles

Les lois de De Morgan, sont des équivalences sémantiques liant la conjonction à la négation d'une disjonction, et inversement.

$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi \qquad \text{(de Morgan)}$$

$$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi \qquad \text{(de Morgan)}$$

L'implication peut se décomposer en une disjonction et une négation. Elle est inversée par négation, et a également une interaction avec la conjonction.

$$\varphi \to \psi \equiv \neg\varphi \vee \psi \qquad \text{(implication)}$$

$$\varphi \to \psi \equiv \neg\psi \to \neg\varphi \qquad \text{(contraposition)}$$

$$(\varphi \wedge \psi) \to \theta \equiv \varphi \to (\psi \to \theta) \qquad \text{(currification)}$$

Il semble raisonnable d'affirmer qu'une variable propositionnelle x et sa négation $\neg x$ ne puissent être toutes deux vraies. Ce résultat constitue le principe de *non-contradiction*. En outre, si x est faux, alors $\neg x$ est vrai, et donc nécessairement l'un parmi x et $\neg x$ doit être vrai (principe du *tiers exclu*).
On peut également vérifier que, si une négation inverse la signification d'une formule, une deuxième négation rétablit la sémantique d'origine.

$$\varphi \wedge \neg\varphi \equiv \bot \qquad \text{(non-contradiction)}$$

$$\varphi \vee \neg\varphi \equiv \top \qquad \text{(tiers exclu)}$$

$$\neg\neg\varphi \equiv \varphi \qquad \text{(double négation)}$$

Enfin, on a les équivalences sémantiques suivantes dont la démonstration est laissée au soin du lecteur.

$$\varphi \wedge \top \equiv \varphi \qquad \text{(élément neutre)}$$

$$\varphi \vee \bot \equiv \varphi \qquad \text{(élément neutre)}$$

$$\varphi \wedge \bot \equiv \bot \qquad \text{(élément absorbant)}$$

$$\varphi \vee \top \equiv \top \qquad \text{(élément absorbant)}$$

$$\varphi \wedge \psi \equiv \psi \wedge \varphi \qquad \text{(commutativité)}$$

$$\varphi \vee \psi \equiv \psi \vee \varphi \qquad \text{(commutativité)}$$

$$(\varphi \wedge \psi) \wedge \theta \equiv \varphi \wedge (\psi \wedge \theta) \qquad \text{(associativité)}$$

$$(\varphi \vee \psi) \vee \theta \equiv \varphi \vee (\psi \vee \theta) \qquad \text{(associativité)}$$

$$\varphi \wedge (\psi \vee \theta) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \theta) \qquad \text{(distributivité)}$$

$$\varphi \vee (\psi \wedge \theta) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \theta) \qquad \text{(distributivité)}$$

1.8 Substitution

La substitution d'une variable propositionnelle x par une formule ψ dans une formule φ consiste à remplacer chaque occurrence de x dans φ par ψ. On la note $\varphi^{\{x \leftarrow \psi\}}$. Par exemple, si $\varphi = (\neg x \vee y) \wedge (\neg x \vee z)$ alors :

$$\varphi^{\{x \leftarrow (x \to y)\}} = (\neg(x \to y) \vee y) \wedge (\neg(x \to y) \vee z)$$

Substitution

On définit la substitution par induction. Dans ces équations, x dénote une variable propositionnelle, et φ, φ_1, φ_2 et ψ des formules.

$$\varphi^{\{x \leftarrow \psi\}} = \varphi \quad \text{(si } x \text{ n'est pas dans } \varphi)$$

$$x^{\{x \leftarrow \psi\}} = \psi$$

$$(\neg\varphi)^{\{x \leftarrow \psi\}} = \neg(\varphi^{\{x \leftarrow \psi\}})$$

$$(\varphi_1 \diamond \varphi_2)^{\{x \leftarrow \psi\}} = \varphi_1^{\{x \leftarrow \psi\}} \diamond \varphi_2^{\{x \leftarrow \psi\}} \quad (\diamond \text{ connecteur binaire})$$

1.9 Formes normales

L'équivalence sémantique montre que des formules logiques peuvent avoir la même signification tout en possédant des syntaxes différentes. Il est alors naturel de se demander s'il existe une écriture *normalisée* permettant de représenter toutes les formules sous une forme commune.

C'est précisément l'objectif des *formes normales*, qui imposent des structures syntaxiques bien définies aux formules logiques.

Formes normales négatives

Considérons la formule

$$\varphi = (x \wedge \neg z) \to (\neg x \wedge \neg(y \wedge \neg z)).$$

En utilisant des équivalences sémantiques, on obtient successivement :

$$\begin{aligned}
\varphi &\equiv \neg(x \wedge \neg z) \vee (\neg x \wedge \neg(y \wedge \neg z)) && \text{(implication)} \\
&\equiv (\neg x \vee \neg\neg z) \vee (\neg x \wedge (\neg y \vee \neg\neg z)) && \text{(de Morgan)} \\
&\equiv (\neg x \vee z) \vee (\neg x \wedge (\neg y \vee z)) && \text{(double négation).}
\end{aligned}$$

Dans cette écriture, la formule ne contient plus que des conjonctions, des disjonctions, des variables propositionnelles et des négations de variables. Cette forme est appelée *forme normale négative*.

Littéral

Un *littéral* est une variable propositionnelle ou la négation d'une variable propositionnelle.

Forme normale négative

Une *forme normale négative* (NNF) est une formule logique ne contenant que les connecteurs $\neg$, $\wedge$ et $\vee$, et dans laquelle la négation ne s'applique qu'aux variables propositionnelles.

Toute formule logique peut être transformée en NNF. La construction repose sur les étapes suivantes :

— éliminer les connecteurs $\rightarrow$ et $\leftrightarrow$ au profit de $\neg$, $\wedge$ et $\vee$;
— propager les négations à l'aide des lois de de Morgan ;
— supprimer les doubles négations.

Formes normales conjonctives

La NNF constitue une première normalisation. On peut aller plus loin en regroupant les littéraux en disjonctions, elles-mêmes combinées par des conjonctions. On obtient alors une *forme normale conjonctive*.

Clause disjonctive

Une *clause disjonctive* est une disjonction de littéraux.

Forme normale conjonctive

Une *forme normale conjonctive* (CNF) est une conjonction de clauses disjonctives.

À partir de la NNF précédente, on obtient une CNF par distributivité :

$$\begin{aligned}\varphi &\equiv (\neg x \vee z) \vee (\neg x \wedge (\neg y \vee z)) \\ &\equiv ((\neg x \vee z) \vee \neg x) \wedge ((\neg x \vee z) \vee (\neg y \vee z)) \\ &\equiv (\neg x \vee z) \wedge (\neg x \vee \neg y \vee z).\end{aligned}$$

Formes normales disjonctives

Une formule peut également être écrite sous *forme normale disjonctive*.

Clause conjonctive

Une *clause conjonctive* est une conjonction de littéraux.

Forme normale disjonctive

Une *forme normale disjonctive* (DNF) est une disjonction de clauses conjonctives.

La CNF précédente peut être transformée en DNF par distributivité :

$$\begin{aligned}\varphi &\equiv (\neg x \vee z) \wedge (\neg x \vee \neg y \vee z) \\ &\equiv (\neg x) \vee (\neg x \wedge \neg y) \vee (\neg x \wedge z) \vee (\neg y \wedge z) \vee (z).\end{aligned}$$

Une autre méthode de construction de la DNF repose sur la table de vérité. Chaque ligne pour laquelle la formule est vraie fournit une clause conjonctive correspondant à la valuation considérée.

Pour la formule

$$(x \wedge \neg z) \to (\neg x \wedge \neg(y \wedge \neg z)),$$

la table de vérité permet d'obtenir la DNF suivante :

$$\varphi = (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z) \\ \vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge z).$$

On constate que les formes CNF et DNF peuvent entraîner une croissance importante du nombre de termes, parfois exponentielle en fonction du nombre de variables. Par exemple, une DNF comportant n clauses conjonctives à deux littéraux se transforme en une CNF à 2^n clauses :

$$(\ell_1 \wedge \ell'_1) \vee \cdots \vee (\ell_n \wedge \ell'_n) \equiv \bigwedge_{\varepsilon_i \in \{\ell_i, \ell'_i\}} (\varepsilon_1 \vee \cdots \vee \varepsilon_n).$$

Représentation des formes normales en OCaml

En OCaml, une CNF ou une DNF est naturellement représentée par une liste de clauses, chaque clause étant une liste de littéraux. Un littéral est codé par un entier naturel non nul pour une variable propositionnelle, et par un entier négatif pour sa négation.

Une clause est une liste de littéraux sans doublons, éventuellement triée par valeur absolue croissante. Une forme normale est enfin décrite par un enregistrement indiquant son type (CNF ou DNF), le nombre de variables et la liste des clauses.

type literal = int type clause = literal list type kind = CNF | DNF type nf = { kind : kind ; nbvars : int ; clauses : clause list }