IA Pas de Soucis!

Un projet de Evrard Maurice,
Lejeune Grégory,
Lejeune Lucas,
Mathieu Louca,
Pluvinage Victor
et de Ralet Vincent.

November 15, 2020

1 Histoire.

1.1 C A VOUS.

2 L'approche logique.

2.1 Les bases de la logique

2.1.1 Introduction à la logique propositionnelle.

L'importance de la logique propositionnelle est immense en mathématiques et en cryptographie, mais également, comme nous allons le voir, en informatique.
Voici une fameuse lapalissade, exemple typique d'utilisation d'une phrase ne découlant de rien d'autre que de cette logique:

"15 minutes avant sa mort, il était encore en vie."[1]

Évidemment, grâce à notre capacité déductionnelle, nous pouvons tous définir cette phrase comme vraie, c'est ici, une vérité dite "de langage".

2.1.2 Un peu de vocabulaire!

La logique propositionnelle possède son propre vocabulaire, il est presque indispensable de connaître son vocabulaire et sa syntaxe afin même de pouvoir en comprendre les concepts.
Tout d'abord, un **langage formel** est un ensemble de mots que l'on peut obtenir en utilisant un alphabet[2], un langage possède plusieurs **lois**, cet ensemble de lois sera appelé la **syntaxe**, qui définissent les différentes manières grâce auxquelles les éléments de l'alphabet (aussi appelés les **symboles**) peuvent se placer pour former quelque chose de cohérent au langage.
Par exemple, en français, vous n'écririez pas "Jème lai vwaturres", mais plutôt, "j'aime les voitures", aussi bête que ce petit exemple puisse paraître, il s'agit là de l'une des nombreuses fois où l'on se plie aux règles d'une grammaire logique.
Un mot respectant toutes les règles de syntaxe sera alors appelé un **mot bien-formé**.
La logique propositionnelle se compose donc d'un langage formel, et de sémantiques donnant du sens aux mots bien formés, répondant au nom de **propositions**.
Les propositions logiques sont désignées par des lettres, comme $A, B, C...$, ou par des lettres indicées comme $A_2, B_4...$ Pour relier ces propositions, on utilise des connecteurs, répertoriés dans le tableau ci-dessous.

[1] https://fr.wikipedia.org/wiki/Lapalissade

[2] Cela peut être un alphabet comme abcd...yz tout comme un alphabet composé uniquement de 1 et de 0.

nom	symbole	autre nom
négation	$\neg$	NOT
conjonction	$\wedge$	AND
disjonction (I)	$\vee$	OR
disjonction (E)	$\oplus$	XOR
implication	$\Rightarrow$	IF..THEN
équivalence	$\Leftrightarrow$	IFF

En plus de ces connecteurs viennent s'ajouter les deux valeurs logiques à la base de tout, Vrai et Faux.

2.1.3 Pour quelques exemples de plus...

Voici quelques exemples d'énoncés de logique propositionnelle dans un cadre assez éloigné des mathématiques, en espérant que cela fasse sens au lecteur. Supposons que A et B soient deux propositions logiques.
A: Je suis boulanger.
B: Je sais faire des gâteaux.
Nous pouvons ainsi relier ces deux propositions avec les connecteurs vus dans le tableau ci-dessus.
$\neg A$: Je ne suis pas boulanger.
$\neg B$: Je ne sais pas faire des gâteaux.
$A \wedge B$: Je suis boulanger et je sais faire des gâteaux.
$A \vee B$: Je suis boulanger ou je sais faire des gâteaux (Les deux propositions peuvent être vraies, tout comme une seule des deux).
$A \oplus B$: Je suis boulanger ou alors je sais faire des gâteaux. (Une seule de ces deux propositions doit être vraie)
$A \Rightarrow B$: Je suis boulanger, donc je sais faire des gâteaux.
$A \Leftrightarrow B$: Je suis boulanger si et seulement si je sais faire des gâteaux.
...

2.1.4 Quelques tables de vérité.

Après tant d'exemples "instructifs", il serait temps de passer aux fameuses **tables de vérité**, ces tables seront d'une importance capitale lors de résolutions de problèmes, les voici donc:

Table 1:
Négation.

A	$\neg A$
F	V
V	F

Table 2:
Conjonction.

A	B	$A \wedge B$
F	F	F
F	V	F
V	F	F
V	V	V

Table 3:
Disjonction (OR).

A	B	$A \vee B$
F	F	F
F	V	V
V	F	V
V	V	V

Table 4:
Disjonction (XOR).

A	B	$A \oplus B$
F	F	F
F	V	V
V	F	V
V	V	F

Table 5:
Implication.

A	B	$A \Rightarrow B$
F	F	V
F	V	V
V	F	F
V	V	V

Table 6:
Équivalence

A	B	$A \Leftrightarrow B$
F	F	V
F	V	F
V	F	F
V	V	V

Les tables 1 à 4 doivent sans doute paraître logique, je m'attarderai toutefois sur la table 5, en effet les deux phrases $F \Rightarrow V$ et $F \Rightarrow F$ sont toutes deux vraies. Cela est du au **principe d'explosion**[3]: Du faux, on peut déduire absolument n'importe quoi! (On verra plus tard que l'on peut noter ceci $A \wedge \neg A \models B$).
Pour ce qui est de la table 6, il faut savoir que certains notent $\Leftrightarrow$ comme étant $\equiv$, cette notation a l'avantage d'accentuer le fait que cette relation n'est autre que la relation d'équivalence.

2.1.5 Résolution de problèmes en logique propositionnelle.

Maintenant que nous avons acquis les bases de la logique propositionnelle, attaquons nous à quelques problèmes.
En voici un premier,

$$\boxed{\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B}$$

Pour résoudre ceci, nous allons utiliser une grande table de vérité:

[3] https://fr.wikipedia.org/wiki/Principe_d%27explosion

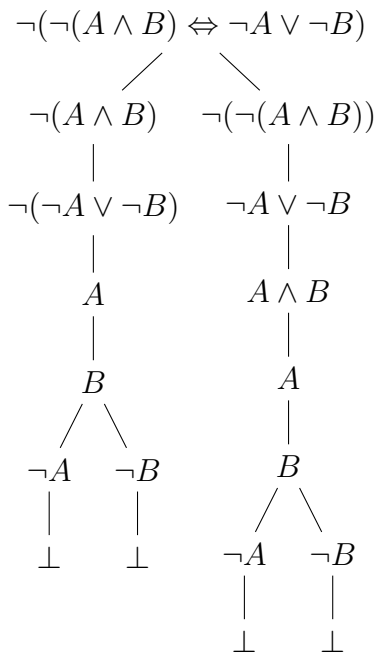
A	B	$A \wedge B$	$\neg(A \wedge B)$	$\neg A$	$\neg B$	$\neg A \vee \neg B$	$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
F	F	F	V	V	V	V	V
F	V	F	V	V	F	V	V
V	F	F	V	F	V	V	V
V	V	V	F	F	F	F	V

Ainsi, comme nous pouvons le constater, la dernière ligne est remplie de "V", cela veut donc dire que nous venons de prouver la relation $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$, aussi connue sous le nom de "la loi de DeMorgan", nous avons prouvé par la même occasion que $\neg(A \wedge B) \Leftrightarrow \neg A \vee B$ est **valide**, cela veut dire qu'elle sera toujours vraie, peu-importe les A, et les B.

Il existe toutefois une autre manière de faire, il s'agit d'utiliser un arbre, cette méthode requiert moins d'étapes et nous permettra de résoudre certains problèmes de manière plus simple. Toutefois, il y a quelques règles à respecter lors de l'utilisation d'un arbre, ces règles sont répertoriées dans le tableau ci-dessous.

$\neg(\neg\phi)$ $\vert$ ϕ	$\phi \wedge \psi$ $\vert$ ϕ $\vert$ ψ	$\neg(\psi \wedge \phi)$ $/ \quad \backslash$ $\neg\psi \quad \neg\phi$
$\phi \vee \psi$ $/ \quad \backslash$ $\phi \quad \psi$	$\neg(\phi \vee \psi)$ $\vert$ $\neg\phi$ $\vert$ $\neg\phi$	$\phi \Rightarrow \psi$ $/ \quad \backslash$ $\neg\phi \quad \psi$
$\neg(\phi \Rightarrow \psi)$ $\vert$ ϕ $\vert$ $\neg\psi$	$\phi \Leftrightarrow \psi$ $/ \quad \backslash$ $\phi \quad \neg\phi$ $\vert \quad \vert$ $\psi \quad \neg\psi$	$\neg(\phi \Rightarrow \psi)$ $/ \quad \backslash$ $\phi \quad \neg\phi$ $\vert \quad \vert$ $\neg\psi \quad \psi$

Le but du jeu avec un arbre logique, c'est de terminer chacune des branches de l'arbre par $\perp$ (C'est le symbole utilisé pour les contradictions). Ainsi, avec un arbre, nous commençons par utiliser l'opposé de notre hypothèse de base (ici, l'opposé de $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$, c'est $\neg(\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B)$), (en général, une contradiction arrive quand nous nous retrouvons avec A et $\neg A$ sur la même branche.)
Passons maintenant à la preuve



Chaque branche de l'arbre fini bien par $\perp$, nous venons donc de prouver la loi de DeMorgan, avec l'aide de notre arbre de démonstration.

Ci dessous, le lecteur pourra s'essayer à la démonstrations de certaines lois logiques célèbres, avec l'aide d'un tableau ou avec un arbre.

$$\neg\neg A \Leftrightarrow A \tag{2.1.1}$$

$$A \wedge \neg A \Leftrightarrow F \tag{2.1.2}$$

$$A \vee \neg A \Leftrightarrow T \tag{2.1.3}$$

$$A \wedge F \Leftrightarrow F \tag{2.1.4}$$

$$A \vee V \Leftrightarrow V \tag{2.1.5}$$

$$\begin{cases} A \vee B \Leftrightarrow B \vee A \\ A \wedge B \Leftrightarrow B \wedge A \end{cases} \tag{2.1.6}$$

$$\begin{cases} A \wedge (B \wedge C) \Leftrightarrow (A \wedge B) \wedge C \\ A \vee (B \vee C) \Leftrightarrow (A \vee B) \vee C \end{cases} \tag{2.1.7}$$

$$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C) \tag{2.1.8}$$

2.2 Les ensembles de propositions.

Les ensembles de propositions, comme leur nom l'indique, sont des ensembles mathématiques, composés de **formules logiques**. Ces formules sont dites soit:

- **consistantes**, signifiant qu'il est possible d'en tirer du Vrai, par exemple $A \wedge B$ ou encore $\neg\neg A \Rightarrow A$.
- **inconsistances**, signifiant que l'on ne peut en tirer que du Faux, par exemple $A \wedge \neg A$, ces formules peuvent être notées $\bot$.
- **valides**, signifiant qu'elles ne peuvent être que Vraies, comme $A \vee \neg A$ (principe du tiers-exclus), une formule **valide** est par définition toujours **consistante**, on appelle bien souvent ces formules valides des **tautologies**, ces formules pourront être notées $\top$.

- **contingentes**, impliquant que l'on peut tirer de la formule du faux, tout comme du vrai, une **tautologie** ne peut pas être formule contingente, un exemple de formule contingente serait $A \vee B$.

Les ensembles aussi ont leur propre terminologie, ainsi, si l'on prend l'ensemble noté S, il pourra être qualifié également de **consistant**, si il n'y a pas de contradictions au sein de S et qu'il n'y a aucune formule inconsistante contenue dans S, autrement S sera défini comme étant **inconsistant**.

Il y a plusieurs manières d'**inférer** quelque chose d'un ensemble logique.
Une première manière est de prendre les **prémisses** qui nous intéressent, et d'écrire

1. **Premisse_A**
2. **Premisse_B**
⋮

n. **Premisse_X**

$\therefore$ **Conclusion**

Prenons un ensemble **consistant** S composé des formules A et $A \Rightarrow B$.
On pourrait alors noter S comme étant $S = \{A, A \Rightarrow B\}$ (ce qui est parfaitement équivalent à écrire $S = A \wedge (A \Rightarrow B)$),
De ceci, nous allons utiliser l'opérateur de la déduction, $\vDash$, ceci nous permettra ainsi écrire $S \vDash A$, littéralement "De S, nous déduisons A".
Ce principe est encore plus flagrant quand nous utilisons la notation suivante : $A, A \Rightarrow B \models A$, où $A, A \Rightarrow B$ n'est autre que l'ensemble S, avec une notation légèrement différente.

Je n'ai pas choisi cet ensemble de manière anodine, car, grâce à celui-ci, nous allons pouvoir utiliser une **règle d'inférence** connue sous le nom du MP (Modus Ponens), le lecteur ne devrait toutefois pas s'inquiéter, un tableau recensant d'autres règles d'inférence sera présenté à la page suivante. Une première manière de noter cette règle d'inférence serait de faire usage de la notation que nous avons vue plus haut.

1. A (De cet ensemble, nous savons A)
2. $A \Rightarrow B$ (De cet ensemble, nous savons que $A \Rightarrow B$)

$\therefore$ B

Une autre manière serait d'utiliser l'opérateur $\vDash$ (celui de la **déduction**), de la manière suivante : $A, A \implies B \vDash B$, nous pourrions même être tentés d'utiliser la **règle d'addition**, disant que si l'on a $S \vDash B$, alors, on peut rajouter la formule B à S. Et ainsi, notre ensemble S de base pourra être réécrit en $S = A, A \Rightarrow B, B$ Et maintenant, comme promis, voici un tableau comprenant toutes les règles d'inférence qui seront bien pratiques pour travailler avec les ensembles logiques.

Règle d'inférence	Tautologie	Nom de la règle d'inférence
$A, B \vDash A \wedge B$	$A \wedge B \Rightarrow A \wedge B$	Loi de combinaison
$A, B \vDash A$	$(A \wedge B) \Rightarrow A$	Loi de la simplification
$A, A \Rightarrow B \vDash B$	$A \wedge (A \Rightarrow B) \Rightarrow B$	Modus Ponens
$\neg B, A \Rightarrow B \vDash \neg A$	$\neg B \wedge (A \Rightarrow B) \Rightarrow \neg A$	Modus Tollens
$A \Rightarrow B, B \Rightarrow C \vDash A \Rightarrow C$	$(A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$	Syllogisme hypothétique
$A \vee B, \neg A \vDash A$	$(A \vee B) \wedge \neg A \Rightarrow B$	Syllogisme disjonctif
$A \Rightarrow B \vDash A \Rightarrow (A \wedge B)$	$(A \Rightarrow B) \Rightarrow (A \Rightarrow (A \wedge B)$	Règle d'absorption
$A \Rightarrow B, C \Rightarrow B, A \vee C \vDash B$	$(A \Rightarrow B) \wedge (C \Rightarrow B) \wedge (A \vee C) \Rightarrow B$	Elimination disjonctive
$A, \neg A \vDash B$	$A \wedge \neg A \Rightarrow B$	Principe d'explosion

Le lecteur pourra s'exercer à démontrer la validité des tautologies présentées dans le tableau-ci dessus en tant qu'exercice.

2.2.1 Le besoin d'algorithme, présentation de l'algorithme de Quine.

Un ensemble de formules, tout comme une formule peut-être inconsistant, cela signifie que notre ensemble est équivalent à F,et comme nous l'avons vu dans le tableau ci-dessus, on peut déduire absolument n'importe quoi d'un ensemble inconsistant.

Le problème des tableaux de vérités, c'est qu'ils prennent de la place, et même, beaucoup de place. C'est pour cela qu'est venu la nécessité de créer des algorithmes de résolution d'ensembles de formules, afin de faciliter le travail des logiciens, et ainsi, de réduire le temps de calcul nécessaire à un ordinateur.

L'algorithme de Quine se déroule en plusieurs étapes, tout d'abord, nous allons simplifier, si possibles, les formules logiques contenues dans notre ensemble de formules, par exemple, au lieu d'écrire $A \vee V$, nous pourrons écrire simplement V, et ainsi, supprimer ce V de notre ensemble, comme $B, V \models B$, il y a bien d'autres simplifications possibles, nous laisserons au lecteur le soin d'en trouver.

Une fois cette première étape passée, il suffit d'utiliser un arbre logique (ou éventuellement une table de vérité) et enfin, nous avons pu prouver des formules logiques avec l'aide de l'agorithme de Quine.

2.3 Une logique? Des logiques!

Les logiciens, non-contents de la seule logique propositionnelle, ont créée un très grand nombre de logique, j'en liste quelques une ci-dessous.

- La logique linéaire, créée par un français, elle est basée sur la gestion de ressources, c'est une des nombreuses logiques n'excluant pas le tiers exclus, en effet, en logique linéaire, $A \vee \neg A \not\vDash V$, cela est simplement du au fait que nous "utilisons" la ressource A une fois, elle n'existe donc plus réellement, et donc $\neg A$ n'existe pas comme A est devenu indéterminé.
- La logique floue, dans laquelle une proposition est vraie selon un certain degré de probabilité.
- La logique de premier ordre, ou logique des prédicats, cette logique sera abordée dans le chapître sur Prolog.
- La logique booléenne, elle est basée sur les portes logiques, les circuits logiques, et les ensembles.
- La logique combinatoire, logique inventée pour formaliser la notion de fonction, et pour limiter le nombre d'opérateurs nécessaires pour définir le calcul des prédicats.
- La logique modale, ayant recours à des opérateurs comme "il est nécessaire que" ou "il est possible que".
- Et bien d'autres...

2.4 Introduction à Prolog et à la logique des Prédicats.

2.5 Room for later.

3 L'approche heuristique.

3.1 P = NP?

3.2 Approche exacte vs Approche heuristique.

3.3 Promenons-nous dans les bois.

3.4 Les bases de Lisp.

3.5 Room for later.

4 Le Machine Learning.

4.1 Définition du Machine Learning

Le Machine Learning, ou Apprentissage Automatique, est un type d'intelligence artificielle qui avec les données à analyser et sur lesquelles s'entraîner permet aux ordinateurs d'apprendre par expérience sans avoir été explicitement programmé à cet effet ou par intervention humaine. Cela consiste en algorithmes d'apprentissage qui améliorent leur performance à exécuter des tâches au fil du temps grâce à de l'expérience.

4.2 Les Maths dans le Machine Leanring.

a. De nombreux data scientists (chargés de la gestion, de l'analyse et de l'exploitation des données au sein d'une entreprise) considèrent le machine Learning comme un apprentissage statistique.

b. Matrice et algèbre matricielle, exemple :

   i. Les suggestions d'amis sur Facebook

   ii. Recommandation de vidéo sur Facebook

   iii. $\cdots$

c. Fonction, variable, équation et graphique, ...

4.3 Les résaux de neurones

Une couche de neurones d'entrées, plusieurs couches cachées, une couche de sortie suivie d'une fonction d'activation. Chaque neurone possède une valeur obtenue par une fonction de combinaison étant la somme des valeurs des neurones de la couche précédente, chacune multipliée par un poids spécifique
$z = x_1 \cdot p_1 + x_2 \cdot p_2 + \cdots + x_n \cdot p_n$
Une fois la (ou les) valeur de la couche de sortie obtenue, on applique à celle-ci une fonction d'activation qui transforme la valeur en fonction d'un seuil. Si en dessous du seuil, inactif (0/-1), aux environs du seuil, phase de transition, et au-dessus du seuil, actif (1/>1). Le type de fonction varie d'un cas à l'autre, mais les plus récurrentes sont la fonction sigmoïde $\frac{1}{1+e^{-x}}$ la fonction

tangente hyperbolique $\frac{2}{1+e^{-2x}} - 1$ ou encore la fonction ReLU $\begin{cases} 0 & x < 0 \\ x & x >= 0 \end{cases}$
Durant l'apprentissage, les poids sont des valeurs prises au hasard. Il faut donc les ajuster pour fournir une réponse qui se rapproche au mieux de la réalité. Comme on entraîne notre réseau, on connaît la vraie valeur finale. On va donc appliquer une fonction de coût afin de calculer le gradient d'erreur entre la valeur réelle et la valeur prédite $\frac{1}{2}(y_r - y_p)^2$, et ainsi mettre à jour les poids par rétropropagation (! il y a des maths plus compliquées derrière). A chaque nouvelles données injectées lors de l'apprentissage, le réseau est plus performant.

4.4 Intro au langage Python

a. Créé par Guido van Rossum au Stichting Mathematisch Centrum en Hollande.

b. Python est le successeur d'un langage de programmation nommé "ABC"

c. Le nom du langage vient de la série Monty Python's flying Circus dont Guido Van Rossum était fan. Cependant l'image du serpent paraissait plus évidente pour tout le monde, il a donc décidé d'utiliser celle-ci comme symbole du langage.

d. La première version de Python paraissait en 1991

e. C'est un langage de programmation open source, c'est à dire gratuit et libre d'utilisation