ÉVALUATION D'UNE EXPRESSION LOGIQUE

Dans le cadre de ce TP nous allons proposer une classe permettant la modélisation et l'évaluation d'une formule de la logique propositionelle classique. Le langage de la logique propositionnelle, introduit par Boole (1854) il y a un peu plus de 150 ans, est une manière naturelle de représenter des connaissances. Avant de décrire le travail qui devra être réalisé, nous décrivons le langage de la logique propositionnelle et sa sémantique.

Syntaxe : le langage propositionnel

Nous commençons par définir les principaux éléments syntaxiques de la logique propositionnelle. Pour spécifier la syntaxe, il est nécessaire de définir les symboles pouvant être utilisés pour formuler des énoncés. Ces symboles forment alors l'alphabet du langage :

Définition 1 *L'alphabet de la logique propositionnelle est constitué :*

- *des symboles* $\bot$ *et* $\top$ *représentant respectivement les constantes propositionnelles "faux" et "vrai" ;*
- *d'un ensemble infini dénombrable de symboles propositionnels, noté PS ;*
- *de l'ensemble des connecteurs logiques usuels : le symbole* $\wedge$ *est utilisé pour la conjonction,* $\vee$ *pour la disjonction,* $\neg$ *pour la négation,* $\Rightarrow$ *pour l'implication,* $\Leftrightarrow$ *pour la bi-implication et* $\oplus$ *pour le ou exclusif ;*
- *des symboles de ponctuation "(" et ")".*

L'ensemble des formules finies formées à partir de cet alphabet est noté *PROP* et est défini inductivement de la manière suivante :

- tout élément de $PS \cup \{\bot, \top\}$ est élément de *PROP* ;
- si $\alpha \in PROP$ alors $(\neg\alpha)$ est élément de *PROP* ;
- si α et β sont des éléments de *PROP* alors $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \Rightarrow \beta)$, $(\alpha \Leftrightarrow \beta)$ et $(\alpha \oplus \beta)$ sont éléments de *PROP*.

Exemple 1 *Soit* $\{a,b,c,d\}$ *un ensemble de symboles propositionnels, alors* $(((a \Leftrightarrow b) \vee (\neg d)) \wedge c)$ *appartient à PROP, contrairement à* $(((a \Leftrightarrow b)$ *et* $((\neg d)\wedge)$.

Il est possible de considérer des opérateurs n-aires. Cependant, pour la suite de ce TP nous ne considérerons que des opérateurs binaires.

Sémantique

La sémantique de la logique propositionnelle associe une signification à ses formules et explique les conditions qui rendent les formules vraies ou fausses. Pour déterminer la valeur de vérité d'une formule il suffit de connaître les valeurs de vérité de ces composantes les plus simples, c'est-à-dire les constantes et les opérateurs. Nous définissons $\bot = faux$, $\top = vrai$ et la sémantique de l'ensemble des opérateurs logiques comme reporté dans le tableau 1.

x	y	$\neg x$	$(x \vee y)$	$(x \wedge y)$	$(x \Rightarrow y)$	$(x \Leftrightarrow y)$	$(x \oplus y)$
vrai	*vrai*	*faux*	*vrai*	*vrai*	*vrai*	*vrai*	*faux*
vrai	*faux*	*faux*	*vrai*	*faux*	*faux*	*faux*	*vrai*
faux	*vrai*	*vrai*	*vrai*	*faux*	*vrai*	*faux*	*vrai*
faux	*faux*	*vrai*	*faux*	*faux*	*vrai*	*vrai*	*faux*

Table 1: Sémantique usuelle des opérateurs logique.

La sémantique permet de définir des règles d'interprétations pour les formules à partir de la valeur de vérité de chacune des propositions qui les composent. Ces règles sont les suivantes :

Définition 2 (Interprétation) *Une interprétation I est une application de PS dans $\mathbb{B}$ qui attribue une valeur de vérité à chaque symbole propositionnel :*

$$I : PS \to \mathbb{B}$$

Soit I une interprétation, la sémantique des propositions selon cette interprétation I est définie inductivement telle que $\forall \alpha, \beta \in PROP$:

- $I(\top) = vrai$ *et* $I(\bot) = faux$ *;*
- $I(\neg\alpha) = vrai$ *si et seulement si* $I(\alpha) = faux$ *;*
- $I(\alpha \circ \beta) = I(\alpha) \circ I(\beta)$ *tel que* $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow, \oplus\}$ *respecte la sémantique énoncée précédemment.*

Comme nous pouvons le voir, l'interprétation d'une formule est définie par l'interprétation des variables qui la composent. Cette interprétation est communément représentée comme l'ensemble des littéraux interprétés à vrai et dans ce cas la sémantique des opérateurs ensemblistes est conservée.

Exemple 2 *Soient $\Sigma = \{(a \vee b) \wedge (a \Rightarrow c)\}$ une formule propositionnelle et $I = \{a, b, \neg c\}$ une interprétation de Σ. Nous avons $I(\Sigma) = I(\top \vee \top) \wedge I(\top \Rightarrow \bot) = \top \wedge \bot = \bot$.*

Représentation et évaluation des formules propositionnelles

Pour représenter une formule propositionnelles en mémoire nous allons utiliser un arbre. En effet, étant donnée la structure arborescente d'une formule de la logique propositionnelle il est évident que cette structure de donnée est la plus adaptée. Comme pour la représentation d'une expression arithmétique nous allons considérer deux types de noeud : les opérateurs et les opérandes. Les opérandes sont les symboles logiques ($\wedge, \vee, \Rightarrow, \Leftrightarrow, \oplus$) tandis que les opérateurs sont les variables propositionnelle ou les constantes $\bot$ et $\top$.

1. Écrivez une classe abstraite `Formule` qui a les méthodes abstraites `eval` qui retourne un booléen et `toString` qui retourne une chaîne de caractères comme méthodes abstraites. Cette classe représentera de manière générale une formule de la logique propositionnelle.
2. Écrivez deux classes abstraites `Operateur` et `Operande` qui héritent de la classe `Formule`.
3. Écrivez la classe `Vrai` qui étend la classe `Operande` et implémente les méthodes `eval` et `toString` telles que la méthode `eval` retourne toujours `true` et la méthode `toString` retourne la chaîne de caractère "V".
4. Écrivez la classe `Faux` qui étend la classe `Operande` et implémente les méthodes `eval` et `toString` telles que la méthode `eval` retourne toujours `false` et la méthode `toString` retourne la chaîne de caractère "F".
5. Écrivez la classe `Variable` qui étend la classe `Operande` et qui a comme attributs une chaîne de caractère *nom* qui représente le nom de la variable et un entier v permettant de représenter la valuation courante de la variable propositionnelle. Cette variable v égale -1 si la variable n'a pas de valeur de vérité, 0 si elle est évaluée à fausse et 1 si elle est évaluée à vraie. Cette classe implémente les méthodes `toString` et `eval` telles que la méthode `toString` retourne la variable *nom* et `eval` retourne `false` si $v = 0$, `true` si $v = 1$ et retourne une exception si $v = -1$ (créer cette exception que vous nommerez `NotEvaluableException`).
6. Écrivez les classes `Ou`, `Et`, `OuExclusif`, `Implique` et `Equivalent` qui héritent de la classe opérateur et qui possèdent deux attributs `op1` et `op2` de type `Formule`. Implémentez convenablement les méthodes `toString` et `eval` en fonction de l'opérateur considéré.
7. Écrivez une classe `Test` pour tester votre programme. L'idéal est de conserver les variables propositionnelles dans un tableau afin de faire varier la valeur de vérité de ces dernières.