Travaux pratiques du cours
Algorithmique et programmation à destination des étudiants d'IMSD
Université de Lorraine, FST

Sujet proposé par Jean Lieber Dernière version : 16 août 2019

Page du cours : `http://homepages.loria.fr/JLieber/cours/algo-prog-M2-IMSD/`

Ce document décrit le sujet de TP du cours d'algorithmique et programmation. Il est organisé comme suit :

1 Objectifs du TP

Un premier objectif est la manipulation d'arbres, plus précisément d'arbres représentant des expressions (voir §2.2.4.2 dans le cours).

Un deuxième objectif est d'apprendre ou de consolider des techniques de programmation orientée objet (notamment : encapsulation, polymorphisme et héritage), sachant que ces notions ne sont pas définies dans ce document, ni dans le cours.

Un troisième objectif est l'apprentissage de notions relatives à la manipulation algorithmique de formules de logique propositionnelle.

2 Cadre du TP : les formules propositionnelles

2.1 Rappel sommaire sur la logique propositionnelle

Le chapitre 2 du cours `http://homepages.loria.fr/JLieber/cours/logiqueL3/` donne une définition plus précise de la syntaxe et de la sémantique de la logique propositionnelle que ce qui suit. On pourra s'y référer.

Une ***formule*** de la logique propositionnelle est une formule s'appuyant sur deux ensembles de symboles :

- Un ensemble dénombrable de ***variables propositionnelles*** qui s'interprètent comme des propositions atomiques ;
- Un ensemble de ***connecteurs*** :
  - Les connecteurs d'arité 0 $\top$ (« top ») qui s'interprète comme une affirmation vraie (dans toutes les interprétations) et $\bot$ (« bottom ») qui s'interprète comme une affirmation fausse (dans toutes les interprétations) ;
  - Le connecteur d'arité 1 $\neg$ (« non », la négation logique) ;
  - Les connecteurs d'arité 2 $\wedge$ (« et », la conjonction), **ou** (« ou », la disjonction), $\Rightarrow$ (« implique », l'implication logique), $\Leftrightarrow$ (« est équivalent à », l'équivalence logique) et $\oplus$ (« ou exclusif »).

Par exemple, la formule

$$a \vee \neg(b \Rightarrow (a \wedge c)) \vee (\neg c \Leftrightarrow b)$$

se lit « a est vrai ou il est faut de dire que b implique que a et c sont vrais ou encore le fait que c soit faux équivaut au fait que b soit vrai ».

Une ***interprétation*** $\mathcal{I}$ est une fonction qui à toute variable propositionnelle a associe une valeur de vérité $\mathcal{I}(a) \in \{\texttt{faux}, \texttt{vrai}\}$. Si on considère un ensemble fini de n variables propositionnelles, il y a 2^n interprétations. La fonction $\mathcal{I}$ est étendue à toute formule propositionnelle de la façon suivante :

$$\mathcal{I}(\top) = \texttt{vrai} \qquad \mathcal{I}(\bot) = \texttt{faux} \qquad \mathcal{I}(\neg\varphi) = \textbf{non}\,\mathcal{I}(\varphi) \qquad \mathcal{I}(\varphi \wedge \psi) = \mathcal{I}(\varphi)\,\textbf{et}\,\mathcal{I}(\psi)$$
$$\mathcal{I}(\varphi \vee \psi) = \mathcal{I}(\varphi)\,\textbf{ou}\,\mathcal{I}(\psi) \qquad \mathcal{I}(\varphi \Leftrightarrow \psi) = \mathcal{I}((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)) \qquad \mathcal{I}(\varphi \oplus \psi) = \mathcal{I}(\neg(\varphi \Leftrightarrow \psi))$$

où φ et ψ sont des formules propositionnelles et où **non**, **et** et **ou** sont les opérations booléennes classiques.

On dit qu'une interpretation $\mathcal{I}$ ***satisfait*** une formule φ si $\mathcal{I}(\varphi) = \texttt{vrai}$. La ***table de vérité*** d'une formule φ à n variables est une table à 2^n lignes, chaque ligne décrivant une interprétation $\mathcal{I}$ et la valeur de vérité $\mathcal{I}(\varphi)$. Les

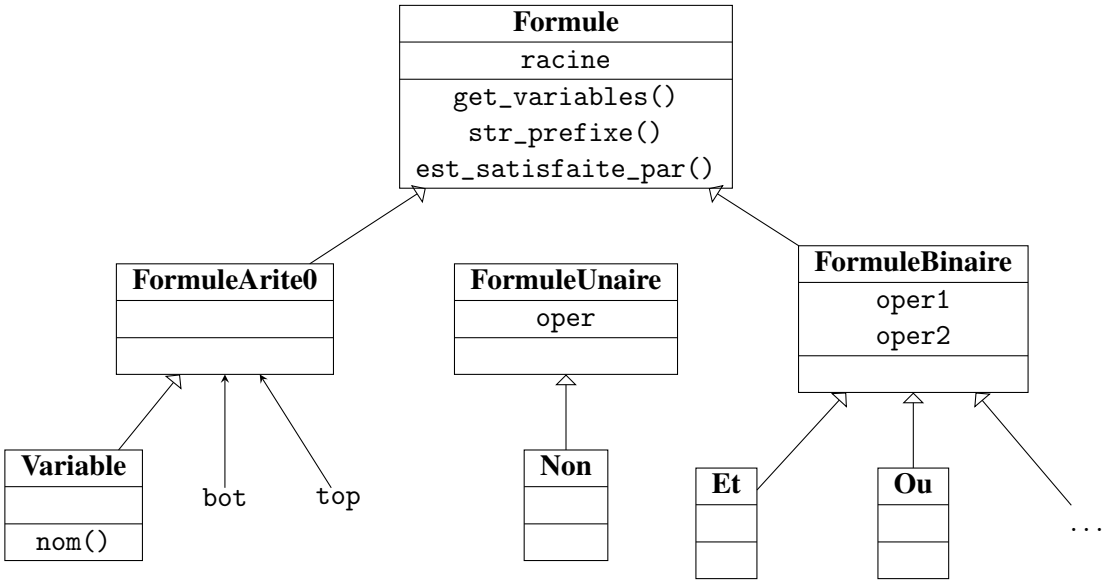

FIGURE 1 – Diagramme de classes des formules propositionnelles (extrait).

autres notions relatives à la logique propositionnelle seront introduites au fur et à mesure des exercices (notamment les notions de formule satisfiable, de tautologie, de conséquence logique $\models$ et d'équivalence logique $\equiv$).

2.2 Classes Python pour décrire et manipuler des formules propositionnelles

Sur la page du cours (`http://homepages.loria.fr/JLieber/cours/algo-prog-M2-IMSD/`) est donné un ensemble de fichiers Python pour décrire les formules propositionnelles selon une approche objet.

Le diagramme de classe des formules spécialisant la classe `Formule` est donné partiellement à la figure 1. Le fichier `TestsFormule.py` regroupe un certain nombre de tests des différentes méthodes et fonctions, en particulier de :

- La méthode `str_prefixe` qui permet l'affihage d'une formule de façon préfixée ;
- La méthode `Parseur.exec()` qui associe à une chaîne de caractères la formule qu'elle représente (quand la chaîne de caractères ne correspond pas à une formule syntaxiquement correcte, il arrive que cela déclenche une exception, mais malheureusement, ce n'est pas systématique) ;
- La méthode `table_verite()` qui permet l'affichage de la table de vérité d'une formule et utilise pour cela la notion d'itérateur sur les interprétations.

3 Exercices introductifs

Après avoir lu (et compris) l'ensemble des fichiers Python proposés, vous effectuerez les deux exercices suivants destinés à comprendre les fonctionnalités du programme.

Ex. 1 Soit $\varphi = (a \oplus b) \Rightarrow \neg a \vee \neg b$.

Représentez la formule φ de deux façons différentes : en utilisant les constructeurs de formules et en utilisant le parseur (l'analyseur grammatical).

Puis utilisez `str_prefixe` pour afficher le résultat.

Ex. 2 Implantez et testez une fonction qui à une formule associe le nombre d'interprétations qui la satisfont. Vous utiliserez pour cela obligatoirement l'itérateur sur les interprétations.

4 Implantation simple (mais complexe en temps de calcul) de quelques inférences déductives

Ex. 3 On dit qu'une formule φ est ***satisfiable*** s'il existe une interprétation qui la satisfait. Écrivez et testez une fonction utilisant un itérateur sur les interprétations pour tester qu'une formule est satisfiable (en vous arrêtant dès que vous avez trouvé une interprétation qui convienne).

Un cas particulier est celui de formules sans aucune variable propositionnelle (les feuilles de l'arbre qui représente une telle formule sont soit de racine $\top$, soit de racine $\bot$). Dans ce cas, il faut introduire une nouvelle variable pour ne pas avoir à considérer un ensemble de variable vide.

Ex. 4 On dit qu'une formule φ est une ***tautologie*** si toutes les interprétations la satisfont. Écrivez et testez une fonction pour tester qu'une formule est une tautologie, en vous appuyant sur le résultat suivant (et sur la fonction définie dans l'exercice précédent) : φ est une tautologie si et seulement si $\neg\varphi$ n'est pas satisfiable.

Ex. 5 Soit φ et ψ deux formules.

On dit que φ ***entraîne*** ψ (noté $\varphi \models \psi$) si toute interprétation qui satisfait φ satisfait aussi ψ.

On dit que φ et ψ sont ***équivalentes*** (noté $\varphi \models \psi$) si pour toute interprétation, elle satisfait φ si et seulement si elle satisfait ψ.

Le théorème de la déduction et son corrolaire indiquent ceci :

$\varphi \models \psi$	si et seulement si	$\varphi \Rightarrow \psi$ est une tautologie
$\varphi \equiv \psi$	si et seulement si	$\varphi \Leftrightarrow \psi$ est une tautologie

En utilisant ces résultats, implantez et testez deux fonctions : une qui teste qu'une formule en entraîne une autre et l'autre qui teste que deux formules sont équivalentes.

Ex. 6 Une ***base de connaissances*** est un ensemble fini de formules. On dit qu'une interprétation ***satisfait*** une base de connaissances B si elle satisfait chaque formule de B. On dit qu'une base de connaissances B ***entraîne*** une formule φ (noté $B \models \varphi$) si pour toute interprétation qui satisfait B, elle satisfait φ.

Implantez la notion de base de connaissances par une classe `BaseConnaissances` (une implantation simple avec un seul attribut sera suffisante, au moins au début ; vous pouvez l'améliorer en utilisant un itérateur).

Implantez et testez la fonction testant qu'une base de connaissances entraîne une formule en vous appuyant sur le résultat suivant : si $B = \{\varphi_1, \varphi_2, \ldots, \varphi_p\}$ alors $B \models \psi$ si et seulement si $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_p \models \psi$.

Ex. 7 On considère ensuite les affirmations suivantes :

(A1) Les seuls mammifères ovipares sont les échidnés et les ornithorynques.

(A2) Un individu ne peut pas être à la fois un échidné et un ornithorynque.

(A3) Toto est un mammifère ovipare qui n'est pas un échidné.

(A4) Toto est un ornithorynque.

Soit $\mathcal{V} = \{e, m, or, ov, t\}$ un ensemble de variables qui sont choisies pour signifier ceci :

e :	être un échidné	ov :	être un ovipare
m :	être un mammifère	t :	être l'individu Toto
or :	être un ornithorynque		

Après avoir formalisé les affirmations (A1) à (A4) par des formules, utilisez la fonction de la question précédente pour tester si l'ensemble des affirmations de (A1) à (A3) est suffisant pour entraîner (A4).

5 Une implantation plus efficace du test de satisfiabilité

Comme on l'a vu à la section précédente, le test de satisfiabilité permet d'implanter plusieurs problèmes d'inférence déductive en logique propositionnelle. L'implantation de la section précédente de ce test est simple, mais peu efficace. En effet, même si ce problème est NP-complet, il en existe des implantations nettement plus efficaces en pratique. Deux algorithmes sont considérés : celui de cette section et celui de la suivante. Vous pouvez les considérer dans l'ordre que vous voulez.

Ex. 8 Soit φ et α, deux formules propositionnelles et a une variable propositionnelle. La substitution de a par α dans φ est la formule propositionnelle obtenue en remplaçant toutes les occurrences de a dans φ par α. Par exemple, la substitution de a par $b \Rightarrow c$ dans $a \wedge (b \vee a)$ est $(b \Rightarrow c) \wedge (b \vee (b \Rightarrow c))$.

Écrivez et testez une méthode pour la substitution d'une variable par une formule dans une formule : il faudra définir cette méthode dans les classes `Formule`, `Variable`, `FormuleUnaire` et `FormuleBinaire`.

Ex. 9 Dans `http://homepages.loria.fr/JLieber/cours/logiqueL3/`, section 2.7.2, un algorithme pour tester la satisfiabilité d'une formule propositionnelle est décrit informellement. Il est inspiré de l'algorithme de Davis-Putnam.

Implantez et testez cet algorithme en considérant un ordre quelconque sur les variables.

Utilisez l'exemple de l'exercice 7 pour estimer le temps de calcul.

Ex. 10 On peut améliorer l'algorithme par un choix approprié de l'ordre des variables. Un critère simple consiste à ordonner les variables par nombre d'occurrences décroissant dans la formule initiale.

Faites une nouvelle version de ce programme en appliquant ce critère pour ordonner les variables et testez6le avec l'exemple de l'exercice 7.

6 Une autre implantation du test de satisfiabilité

L'implantation proposée ici est similaire à l'application de la méthode des tableaux sémantiques à la logique propositionnelle (`http://homepages.loria.fr/JLieber/cours/logiqueL3/`, section 2.7.4), quoiqu'un peu moins efficace. Elle est décomposée en plusieurs exercices.

Ex. 11 Un ***littéral*** est une formule de la forme a (***littéral positif***) ou de la forme $\neg a$ (***littéral négatif***) où a est une variable propositionnelle. On admettra aussi $\top$ et $\bot$ parmi les littéraux pour simplifier. Une formule en ***forme normale disjonctive*** (***FND*** ou ***DNF***, en anglais) est une formule s'écrivant comme une disjonction de conjonctions de littéraux. Par exemple, $(a \wedge \neg b) \vee (a \wedge c) \vee \neg c \vee d$ est une formule sous forme FND avec 4 disjonctions de conjonctions de littéraux. On peut montrer que toute formule peut se mettre sous forme FND : pour toute formule φ, il existe une formule φ_{FND} telle que $\varphi_{\text{FND}} \equiv \varphi$.

Techniquement, pour mettre une formule sous forme FND, on applique à cette formule et à ses sous-formules les équivalences suivantes orientées de gauche à droite, tant que c'est possible :

$$\alpha \oplus \beta \equiv \neg(\alpha \Leftrightarrow \beta) \qquad \alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha) \qquad \alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$$

$$\neg\top \equiv \bot \qquad \neg\bot \equiv \top$$

$$\neg\neg\alpha \equiv \alpha \qquad \neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta \qquad \neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$

$$\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \qquad (\alpha \vee \beta) \wedge \gamma \equiv (\alpha \wedge \gamma) \vee (\beta \wedge \gamma)$$

pour toutes formules α, β et γ.

On peut également utiliser les équivalences suivantes, qui ne sont pas indispensables, mais conduisent à des simplifications :

$$\bot \wedge \alpha \equiv \bot \qquad \alpha \wedge \bot \equiv \bot \qquad \bot \vee \alpha \equiv \alpha \qquad \alpha \vee \bot \equiv \alpha$$

$$\top \wedge \alpha \equiv \alpha \qquad \alpha \wedge \top \equiv \alpha \qquad \top \vee \alpha \equiv \top \qquad \alpha \vee \top \equiv \top$$

Par exemple, on a les équivalences successives pour mettre la formule $a \oplus (b \vee \neg c)$ sous FND (chaque étape ci-dessous correspond à une ou plusieurs applications des équivalences ci-dessus) :

$$\begin{aligned} a \oplus (b \vee \neg c) &\equiv \neg(a \Leftrightarrow (b \vee \neg c)) \equiv \neg((a \Rightarrow (b \vee \neg c)) \wedge ((b \vee \neg c) \Rightarrow a)) \\ &\equiv \neg((\neg a \vee b \vee \neg c) \wedge (\neg(b \vee \neg c) \vee a)) \equiv \neg((\neg a \vee b \vee \neg c) \wedge ((\neg b \wedge \neg c) \vee a)) \\ &\equiv \neg(\neg a \vee b \vee \neg c) \vee \neg((\neg b \wedge \neg c) \vee a) \equiv (\neg\neg a \wedge \neg b \wedge \neg\neg c) \vee (\neg(\neg b \wedge \neg c) \wedge \neg a) \\ &\equiv (a \wedge \neg b \wedge c) \vee ((b \vee c) \wedge \neg a) \equiv (a \wedge \neg b \wedge c) \vee (b \wedge \neg a) \vee (c \wedge \neg a) \end{aligned}$$

Notez que comme $\vee$ et $\wedge$ sont associatives modulo l'équivalence logique (p. ex., $(\alpha \vee \beta) \vee \gamma \equiv \alpha \vee (\beta \vee \gamma)$), les parenthèses entre disjonctions ou entre conjonctions (mais pas les deux à la fois !) sont enlevées.

Implantez et testez un algorithme qui met une formule quelconque sous FND en utilisant toutes les transformations associées aux équivalences ci-dessus.

Ex. 12 Une conjonction de littéraux est insatisfiable si et seulement si elle contient $\bot$ ou elle contient deux littéraux sur la même variable x, l'un étant positif et l'autre négatif (x et $\neg x$, respectivement). Par exemple, $(\bot \wedge a \wedge \neg b)$ et $(a \wedge b \wedge \neg a \wedge \top)$ sont insatisfiables alors que $(a \wedge \neg b \wedge c \wedge \top)$ est satisfiable.

Une formule sous forme DNF est satisfiable si au moins une des conjonctions de littéraux qui la constitue est satisfiable. Ainsi $(\bot \wedge a \wedge \neg b) \vee (a \wedge b \wedge \neg a \wedge \top) \vee (a \wedge \neg b \wedge c \wedge \top)$ est satisfiable.

Utilisez ces résultats pour écrire et tester un programme vérifiant si une formule propositionnelle quelconque est satisfiable ou non.

Utilisez l'exemple de l'exercice 7 pour estimer le temps de calcul.