TP 1 : logique propositionnelle et *model checking*

Information	Valeur
Auteur	Sylvain Lagrue (sylvain.lagrue@utc.fr)
Licence	Creative Common CC BY-SA 3.0
Version document	2.4.0

Exercice : génération automatique de tables de vérités

On souhaite créer un programme qui, étant donné une formule donnée en argument et un ensemble de variables propositionnelles affiche sa table de vérité. On se limitera aux connecteurs ∧ (`and`), ∨ (`or`) et ¬ (`not`). On utilisera pour cela Python 3.6 (ou supérieur).

Exemple de sortie possible

```
> table("(A or B) and not(C)", ["A", "B", "C"])
formule : (A or B) and not(C)
+---+---+---+-------+
| A | B | C | eval. |
+---+---+---+-------+
| F | F | F |   F   |
| T | F | F |   T   |
| F | T | F |   T   |
| T | T | F |   T   |
| F | F | T |   F   |
| T | F | T |   F   |
| F | T | T |   F   |
| T | T | T |   F   |
+---+---+---+-------+
```

Question 1

Écrire une fonction `decomp(n: int, nb_bits: int) -> List[bool]` qui, étant donné un nombre `n` , calcule la décomposition binaire en `nb_bits` de `n` . L'ordre des bits (croissant ou décroissant) est sans importance.

Exemple

```
> decomp(3, 4)
[True, True, False, False]
```

Question 2

Une interprétation peut être vue comme un dictionnaire qui associe à chaque variable propositionnelle la valeur `True` ou la valeur `False`.

Créer une fonction `interpretation(voc: List[str], vals: List[bool]) -> Dict[str, bool]` qui, étant donné un tableau de chaîne de caractère représentant les variables propositionnelles et un liste de valeurs prises par ces variables renvoie une interprétation.

Exemple

```
> interpretation(["A", "B", "C"],[True, True, False])
{"A": True, "B": True, "C": False}
```

Question 3

Créer un générateur d'interprétations `gen_interpretations(voc: List[str]) -> Generator[Dict[str, bool], None, None]` qui, étant donné la liste des variables booléennes, génère une à une les interprétations.

Exemple

```
> g = gen_interpretations(["A", "B", "C"])
> print(next(g))
{"A": False, "B": False, "C": False}
> print(next(g))
{"A": True, "B": False, "C": False}

> for i in gen_interpretations(["toto", "tutu"]):
>     print(i)
{'toto': False, 'tutu': False}
{'toto': True, 'tutu': False}
{'toto': False, 'tutu': True}
{'toto': True, 'tutu': True}
```

Question 4

Créer une fonction `valuate(formula: str, interpretation: Dict[str, bool]) -> bool:` prenant en entrée une chaîne de caractères représentant la formule à évaluer et une interprétation et qui renvoie `True` ou `False`. On utilisera pour cela **exceptionnellement** la fonction `eval`.

Exemple

```
> valuate("(A or B) and not(C)", {"A": True, "B": False, "C": False})
True
```

Question 5

Écrire une fonction permettant d'afficher la table de vérité étant donné une formule propositionnelle et le vocabulaire afférent.

Question 6

Écrire **3** fonctions différentes permettant respectivement de savoir si une formule est valide, contradictoire ou contingente. Tester ces fonctions sur des formules comprenant 1, 2, 3, 5, 10, 15, 20, 25, 30, 50 et 100 variables. Qu'en déduire ?

Question 7

Écrire une fonction `is_cons(f1: str, f2: str, voc: List[str]) -> bool` qui renvoie `True` si la formule `f2` est la conséquence logique de la formule `f1`, `False` sinon.