Travaux Pratiques

Logique Propositionnelle et Algorithme de Quine - Deuxieme essai

Informatique

Année académique 2024-2025

Clément Rouvroy

2 août 2025

Table des matières

1 Consignes

Vous devez coder en OCaml. Votre code doit compiler, si il ne compile pas vous n'aurez aucun point. Vous pouvez utiliser le bouton Executer du site pour voir si votre code compile.

2 Représentation des formules logiques en OCaml

Une formule propositionnelle est construite à partir de variables propositionnelles (des string, comme $P, Q, R, \dots$) et de connecteurs logiques (NON, ET, OU, IMPLIQUE, EQUIVALENT). Pour faciliter l'implémentation de l'algorithme de Quine, nous inclurons également des constructeurs pour les constantes VRAI et FAUX.

Question 1. Ecrire un type OCaml formule qui permet de manipuler les formules.

Question 2. En utilisant le type `formule` ci-dessus, représentez en OCaml les formules logiques suivantes :

1. $P \vee Q$
2. $P \wedge (Q \vee \neg R)$
3. $(A \iff B) \wedge \text{VRAI}$

Nommez vos variables OCaml `exemple1`, `exemple2`, et `exemple3` respectivement.

Question 3. En logique, est-ce que $(A \vee B) \vee C \equiv (C \vee A) \vee B$?

Question 4. En logique, est-ce que $(A \vee B) \wedge C \equiv (A \wedge C) \vee (B \wedge C)$?

3 Fonctions de base sur les formules

Question 5. Écrire une fonction OCaml `hauteur : formule -> int` qui calcule la hauteur d'une formule. Testez votre fonction sur les exemples que vous avez ecrit plus haut. Ecrivez en commentaire les resultats que vous obtenez.

Question 6. Écrire une fonction OCaml `compte_vrai : formule -> int` qui compte le nombre d'occurences de `Vrai` dans une formule.

Question 7. Écrire une fonction OCaml `substituer : formule -> string -> bool -> formule`. Cette fonction prend une formule `f`, un nom de variable `var_nom`, et une valeur booléenne `valeur_bool`. Elle renvoie une nouvelle formule où toutes les occurrences de la variable `Var var_nom` dans `f` sont remplacées par `Vrai` si `valeur_bool` est `true`, et par `Faux` si `valeur_bool` est `false`.

Question 8. Écrire une fonction OCaml `eval_const_formule : formule -> bool`. Cette fonction prend une formule qui est supposée ne contenir que des constructeurs `Vrai`, `Faux` et des connecteurs logiques (c'est-à-dire, plus aucune `Var`). Elle évalue cette formule et renvoie sa valeur de vérité booléenne. Si elle rencontre un `Var`, elle peut lever une exception (par exemple, `failwith "Variable non substituée"`).

4 Algorithme de Quine (sans simplification explicite)

L'algorithme de Quine est une méthode pour déterminer si une formule propositionnelle est une tautologie (toujours vraie) ou satisfiable (peut être vraie). Il fonctionne de manière récursive :

1. Si la formule ne contient plus de variables, on l'évalue (elle doit être composée uniquement de `Vrai`, `Faux` et de connecteurs). Le résultat de cette évaluation est le résultat pour cette branche.
2. Sinon, on choisit une variable X dans la formule.
3. On crée deux nouvelles formules : $F_1 = F[X := \text{VRAI}]$ (où X est remplacée par `Vrai`) et $F_2 = F[X := \text{FAUX}]$ (où X est remplacée par `Faux`).
4. Pour vérifier si F est une **tautologie**, on vérifie si F_1 ET F_2 sont des tautologies.
5. Pour vérifier si F est **satisfiable**, on vérifie si F_1 OU F_2 est satisfiable.

Les fonctions `substituer` et `eval_const_formule` des questions précédentes seront utiles ici.

Question 9. Écrire une fonction OCaml `est_tautologie_quine : formule -> bool` qui implémente l'algorithme de Quine (sans simplification explicite au-delà de la substitution) pour déterminer si une formule est une tautologie. La fonction utilisera `variables_libres`, `substituer`, et `eval_const_formule`.

Question 10. Écrire une fonction OCaml `est_satisfiable_quine : formule -> bool` qui implémente l'algorithme de Quine (sans simplification explicite) pour déterminer si une formule est satisfiable.

5 Algorithme de Quine avec simplification

L'algorithme de Quine peut être rendu plus efficace en simplifiant les formules obtenues après chaque substitution. Par exemple, si on substitue X par `Vrai` dans $X \wedge Y$, on obtient $\text{VRAI} \wedge Y$, qui se simplifie en Y.

Quelques règles de simplification :

— $\neg\text{VRAI} \equiv \text{FAUX}$
— $\neg\text{FAUX} \equiv \text{VRAI}$
— $\neg(\neg A) \equiv A$
— $A \wedge \text{VRAI} \equiv A$
— $A \wedge \text{FAUX} \equiv \text{FAUX}$
— $A \vee \text{VRAI} \equiv \text{VRAI}$
— $A \vee \text{FAUX} \equiv A$
— $A \implies \text{VRAI} \equiv \text{VRAI}$
— $\text{VRAI} \implies A \equiv A$
— $A \implies \text{FAUX} \equiv \neg A$
— $\text{FAUX} \implies A \equiv \text{VRAI}$
— $A \iff \text{VRAI} \equiv A$
— $A \iff \text{FAUX} \equiv \neg A$
— Si $A \equiv B$ (structurellement après simplification), alors $A \iff B \equiv \text{VRAI}$.

Question 11. Écrire une fonction OCaml `simplifier_formule : formule -> formule`. Cette fonction applique une passe des règles de simplification ci-dessus (et d'autres que vous jugerez utiles, comme $A \wedge A \equiv A$). Elle doit simplifier les opérandes d'abord, puis appliquer les règles au niveau actuel. Si une simplification est appliquée (par exemple, `Et(Vrai, f)` devient `f`), la fonction devrait tenter de simplifier le résultat à nouveau. L'objectif est d'obtenir une formule aussi simple que possible en une "passe complète récursive".

Question 12. Modifier la fonction `est_tautologie_quine` en une nouvelle fonction `est_tautologie_quine_simplifiee : formule -> bool`. Après chaque substitution d'une variable par `Vrai` ou `Faux`, la formule résultante doit être passée à `simplifier_formule`. Si la formule se simplifie en `Vrai`, la branche correspondante de l'algorithme de Quine s'arrête et renvoie `true`. Si elle se simplifie en `Faux`, elle renvoie `false`. Sinon, l'algorithme continue sur la formule simplifiée.

Question 13. De même, modifier `est_satisfiable_quine` en `est_satisfiable_quine_simplifiee : formule -> bool` pour intégrer l'étape de simplification.

Question 14. Discutez brièvement des avantages de l'utilisation de la simplification dans l'algorithme de Quine. Quels sont les impacts attendus sur les performances (temps de calcul, nombre d'appels récursifs) ?