TRAVAUX PRATIQUES XXV

Logique et algorithme de Quine

Dans ce sujet, on s'intéresse au problème SAT pour une formule quelconque, construite à partir de constantes, de variables et des connecteurs introduits dans le cours. Pour simplifier, on enlève le connecteur $\leftrightarrow$ de nos formules (il est logiquement équivalent à deux $\rightarrow$ et un $\wedge$).

A Formules propositionnelles

Premières manipulations

On considère des formules propositionnelles définies par le type suivant :

quine.ml
```
type prop =
  | Bot
  | Top
  | Var of int (* l'entier i >= 0 représente la variable propositionnelle x_i *)
  | Non of prop
  | Et of prop * prop
  | Ou of prop * prop
  | Imp of prop * prop
```

0. Écrivez une fonction `taille : prop -> int` qui détermine la taille d'une formule propositionnelle $|.|$.

Pour les tests, vous pourrez utiliser la proposition suivante, donnée en OCaml :

```
let phi = Et(Imp(Et(Var 0, Var 2), (Et(Var 0, Non (Ou(Var 1, Non (Var 2))))), Var 0)
```

1. Quelle formule propositionnelle φ la variable `phi` représente-t-elle ? Écrivez-là "normalement" **avec une feuille et un crayon** avec les connecteurs $\wedge, \vee, \rightarrow, \neg$ et les variables propositionnelles $x_i \in \mathscr{V}$.

2. Écrire une fonction `print_formule : prop -> unit` qui affiche une proposition sous forme infixe. Le parenthésage doit être suffisant pour qu'il n'y ait pas d'ambiguïté (il peut être excessif).

Par exemple, dans un terminal comme utop, on peut obtenir :

Terminal
```
print_formule (Ou (Et (Var 1, Bot), Non (Var 0)));;
((x1 et Faux) ou non x0)- : unit = ()
```

3. Vérifiez votre réponse à la question 1 en affichant `phi`.

Évaluation d'une formule

On introduit le type suivant pour représenter une valuation :

```
type valuation = bool array
```

Ainsi, la valeur logique associée à la variable x_i par une valuation `v : valuation` est `v[i]` .

4. Écrire une fonction `eval : prop -> valuation -> bool` telle que `eval phi v` renvoie eval_v(`phi`). On pourra supposer sans vérification que le tableau `v` est de longueur suffisante pour contenir toutes les variables apparaissant dans `phi`.

B SAT-solvers

Algorithme de Quine

Pour mettre en oeuvre l'algorithme de Quine (vu en cours), on a tout d'abord besoin de pouvoir substituer une variable par $\top$ ou $\bot$ selon la valeur qu'on lui attribue dans une valuation. Plus généralement, on s'intéresse à la substitution d'une variable par une formule.

5. Écrire une fonction `substitue : prop -> int -> prop -> prop` telle que si `phi` et `f` sont des formules propositionnelles, alors `substitue phi i f` renvoie la proposition `phi`$[$`f`$/x_i]$.

6. Écrire une fonction `simplifie : prop -> prop` qui prend en argument une formule propositionnelle et la simplifie avec les règles de l'algorithme de Quine. (Après un appel à `simplifie` , `phi` ne doit plus contenir aucune constante, ou être elle-même réduite à une seule constante.)
   Aide : Cette fonction correspond à une grosse disjonction de cas, il est donc normal que vous utilisiez plusieurs filtrages. Vous pouvez au choix faire une grosse fonction, ou utiliser des fonctions auxiliaire (une par cas, par exemple).

Il ne reste plus qu'à décider quelle variable de φ, à chaque étape, on veut choisir de mettre à Vrai ou Faux.

7. Écrire une fonction `var_max` qui renvoie l'indice maximal d'une variable d'une formule.

8. Écrire une fonction récursive `sat_quine : prop -> bool` qui implémente l'algorithme de Quine décrit dans le cours.
   Aide : Avec tout ce qui a été fait avant, cette fonction est très simple à écrire en version récursive. Pensez à utiliser || en OCaml.
   Variante étoile : trouver une manière efficace de ne pas faire un appel à `var_max` *à chaque fois.*

Pour tester votre fonction, voici un exemple de formule qui n'est pas satisfiable :

$$\varphi_\bot = (x_0 \vee \neg x_1) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_0) \wedge (x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$$

On a obtenu un SAT-solver[1] qui répond à la question de la satisfiabilité, mais on aimerait de plus avoir accès à la valulation qui permet de satisfaire notre formule, dans le cas où elle est satisfiable.
Dans le cadre de l'algorithme de Quine, il s'agit de remonter la branche qui a mené à $\top$. Il faut donc retenir les choix qu'on a fait pour chaque variable.

9. Modifiez votre fonction `sat_quine` en fonction `trouve_valuation` qui renvoie un modèle de la formule prise en entrée si elle est satisfiable, ou lève l'exception `Not_found` sinon.
10. Modifiez-la encore en `trouve_toutes_valuations` une fonction qui renvoie la liste de *tous* les modèles de la formule prise en entrée.

1. L'algorithme de Quine n'est pas le plus utilisé en pratique pour implémenter un SAT-solver. Il s'agit plutôt de l'algorithme DPLL (Davis-Putnam-Logemann-Loveland) qui fonctionne sur le principe du backtracking en essayant de détecter les littéraux dont la valeur est forcée.