Feuille de TD n°8 : logique

MPSI option informatique

Juin 2025

satisfiabilité d'une formule propositionnelle.

Une formule propositionnelle est construite à l'aide de constantes propositionnelles, de variables propositionnelles et de connecteurs logiques.

Les connecteurs logiques seront notés $\neg$ (négation), $\wedge$ (conjonction), $\vee$ (disjonction). On n'en considèrera pas d'autre. Dans cette partie, on étudie le problème de satisfiabilité d'une formule.

Le problème CNF-SAT est défini de la façon suivante. Étant donné une formule sous forme normale conjonctive, admet-elle un modèle, c'est-à-dire une valuation des variables, qui rende la formule vraie ? On souhaite écrire un programme qui teste si une valuation donnée rend une telle formule vraie.

Dans cette partie, on considère que si une formule contient n variables propositionnelles, elles seront désignées par $x_0, x_1, \ldots, x_{n?1}$.

On définit le type OCaml suivant :

```
type clause = Var of int | Non of int | Ou of clause * clause
```

L'argument du constructeur `Var` correspond au numéro de la variable concernée. On notera que cette définition d'une clause ne tient pas compte de l'associativité de la disjonction.

Une formule sous forme normale conjonctive ayant m clauses sera implémentée par une liste de m termes de type `clause`.

Les tableaux seront implémentés par le module Array dont les éléments suivants pourront être utilisés : - `type 'a array`
- création d'un tableau : `make : int -> 'a -> 'a array`
- accès à l'élément d'indice i du tableau `t` : `t.(i)`
- modification de l'élément placé à l'indice i du tableau `t` : `t.(i) <- v`
- taille du tableau : `length : 'a array -> int`

1) Donner le code `Ocaml` correspondant à la clause $c = (x_0 \vee x_1) \vee \neg x_2)$.

2) Donner un code `Ocaml` permettant de définir la formule : $f = (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$.

3) Écrire une fonction de signature `evalue_clause : clause ? bool array ? bool` qui prend en paramètre une clause et une valuation représentée par un tableau contenant à l'indice i, la valeur de vérité de la variable x_i, et renvoie la valeur de vérité de la clause.

4) Écrire une fonction de signature `'evalue_FNC : clause list -> bool array -> bool` qui prend en paramètre une clause et une valuation représentée par un tableau contenant à l'indice i, la valeur de vérité de la variable x_i et évalue une formule donnée sous forme normale conjonctive.

5) Quel résultat obtient-on avec la formule f et le tableau de valuations `[|false;true;true|]` ? Justifier.

6) On souhaite énumérer toutes les valuations possibles pour un nombre de variables fixé. Étant donné une valuation, on considérera que si la valeur true correspond à 1 et la valeur false

correspond à 0, la valuation suivante correspond à l'ajout de 1 au nombre binaire associé. Ainsi, la valuation suivante de `[|false;true;false|]` est `[|false;true;true|]`.
On considère que la valuation suivante de `[|true;true;true|]` n'existe pas.
Écrire une fonction de signature `suivant : bool array -> bool` qui prend en paramètre un tableau de booléens, et et renvoie false s'il n'existe pas de valuation suivante, et true sinon. De plus, dans ce dernier cas, cette fonction devra avoir pour effet de bord de modifier son premier argument en le transformant en la valuation suivante.

7) En déduire une fonction de signature `satisfiable : clause list -> int -> bool` qui prend en paramètre une formule en forme normale conjonctive, son nombre de variables, et renvoie `true` s'il existe une valuation qui rend la formule vraie, `false` sinon.