Conversion en forme normale conjonctive

Florian Bourse

Nous utiliserons les définitions suivantes pour représenter les formules logiques :

Types des formules propositionnelles et des formes normales conjonctives

```
type binary_operator = And | Or | Imp | Eq
type formula =
  | Top
  | Bot
  | Var of int
  | Not of formula
  | Bin of binary_operator * formula * formula

let f1 = Bin(Imp, Bin(Or, Var 1, Var 2), Bin(Or, Var 3, Var 2))
let f2 = Bin(Or, Bin(Imp, Var 1, Var 2), Not(Var 1))
let f3 = Bin(And, Bin(Or, Bin(Or, Var 1,Var 2), Not(Var 3)), Var 4)
let f4 = Bin(Or, Var 1, Bin(And, Var 2, Bin(Or, Not(Var 3), Var 4)))

type litteral = int
(* l'entier n >= 0 représente Var(n) *)
(* l'entier n < 0 représente Not(Var(-n-1)) *)
type clause = int list
type cnf = clause list

let cnf1 = [[-2; 3; 2]]
let cnf2 = [[-2; 2]]
let cnf3 = [[1; 2; -4]; [4]]
let cnf4 = [[1; 2]; [1; -4; 4]]
```

0 Écrire une fonction `nnf (f : formula) : formula` qui prend en entrée une formule logique et qui renvoie une formule équivalente ne contenant pas les constructeurs `Imp` ni `Eq`.

1 Écrire une fonction `morgan (f : formula) : formula` qui utilise les lois de De Morgan et le tiers exclu pour ne plus utiliser le constructeur `Not`. On remplacera les occurences de `Not (Var n)` par `Var (- n - 1)`.
On pourra supposer que cette fonction est toujours utilisée après `nnf`

2 Écrire une fonction `cnf_aux (f : formula) : cnf` qui construit une forme normale conjonctive d'une formule logique. On pourra supposer que cette fonction est toujours utilisée après `nnf` et `morgan`.

3 En déduire une fonction `cnf (f : formula) : cnf` qui permet de construire une formule sous forme normale conjonctive équivalente à une formule donnée.

4 Simplifier les formules sous forme normale conjonctive pour que chaque variable n'apparaisse qu'au plus une fois par clause.