Problema A - Representação de funções booleanas sob a forma de fórmulas de lógica proposicional FND/FNC

DI-UBI | Lógica Computacional | 2021/2022

Uma função booleana é uma função cujos argumentos e resultado são valores pertencentes a um conjunto de dois elementos (representados geralmente por $\{F, V\}$ ou $\{0, 1\}$). Estas assumem a forma $f : \{0, 1\}^k \to \{0, 1\}$, onde k é um inteiro não-negativo que indica o número de argumentos da função (aridade). Para um $k = 0$, a "função" é um elemento constante de $\{0, 1\}$.

Qualquer função booleana com k argumentos pode ser expressa sob a forma de uma fórmula de lógica proposicional com k literais $(x_1, \ldots, x_k)$.

Dadas duas fórmulas de lógica proposicional, φ e ψ, $\varphi \leftrightarrow \psi$ se e apenas se φ e ψ representarem a mesma função booleana.

Uma função booleana pode também ser representada sob a forma de uma tabela de verdade que lista explicitamente o seu valor para todos os valores possíveis dos seus argumentos.

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Tabela 1: Tabela de verdade para uma função booleana f com 3 argumentos.

Problema

O objetivo deste problema é, dada uma função booleana sob a forma de uma tabela de verdade que a representa, obter duas proposições da lógica proposicional equivalentes a essa função, uma em Forma Normal Conjuntiva (FNC) e outra em Forma Normal Disjuntiva.

Forma Normal Conjuntiva

Uma fórmula diz-se em forma normal conjuntiva se for da forma:

$$(\alpha_{1_1} \vee \ \ldots \ \vee \alpha_{1_{k_1}}) \wedge \ldots \wedge (\alpha_{n_1} \vee \ \ldots \ \vee \alpha_{n_{k_n}})$$

onde cada α_{i_j} é um literal.

Cada conjunto de disjunções (disjuntório $\bigvee_{j=1}^{k} \alpha_{i_j}$) é também denominado de cláusula.

Forma Normal Disjuntiva

Uma fórmula diz-se em forma normal disjuntiva se for da forma:

$$(\alpha_{1_1} \wedge \ \ldots \ \wedge \alpha_{1_{k_1}}) \vee \ldots \vee (\alpha_{n_1} \wedge \ \ldots \ \wedge \alpha_{n_{k_n}})$$

onde cada α_{i_j} é um literal.

Cada conjunto de conjunções ($\bigwedge_{j=1}^{k} \alpha_{i_j}$) é também denominado de conjuntório.

Input

Primeira linha: valor k ($0 < k \leq 12$) que indica o número de variáveis da função booleana f.

2^k **linhas seguintes:** cada linha contém $k+1$ valores booleanos (0 ou 1) separados por um espaço, que correspondem aos valores das variáveis x_1 a x_k da função f e o respetivo valor da função para essa combinação de valores (no fundo, é uma linha da tabela de verdade da função f).

Input exemplo

```
3
0 0 0 1
0 0 1 0
0 1 0 0
```

```
0 1 1 1
1 0 0 0
1 0 1 0
1 1 0 1
1 1 1 1
```

(Corresponde à função booleana representada na Tabela 1.)

Output

Primeira linha: string com a fórmula equivalente à função booleana f em FND.

Segunda linha: string com a fórmula equivalente à função booleana f em FNC.

Para formar as fórmulas, deverá associar a cada variável x_i de f uma letra minúscula do alfabeto, começando com $x_1 \to a$ até $x_{12} \to l$.

Deverá ser usado tipo `formula` e a função `print_formula` (incluídas abaixo) para construir e imprimir as fórmulas geradas, sob pena de a solução ser rejeitada pelo Mooshak caso use outro método.

Output exemplo

```
((!a & b & c) | (!a & !b & !c) | (a & b & c) | (a & b & !c))
((a | b | !c) & (a | !b | c) & (!a | b | c) & (!a | b | !c))
```

Sugestão de resolução

É possível obter uma fórmula de lógica proposicional na forma FND equivalente a uma função booleana f por observação da tabela de verdade dessa função. Cada linha dessa tabela onde f tem valor 1 corresponde a um conjuntório da fórmula FND, onde a cada variável x_i fazemos corresponder o literal l_i ou a sua negação ($\neg l_i$), consoante o valor dessa variável seja 1 ou 0, respetivamente. Por outras palavras, o conjuntório obtido a partir de uma linha da tabela de verdade onde f tem valor 1 é definido da seguinte forma:

$$\bigwedge_{i=1}^{k} \begin{cases} l_i, & \text{se } x_i = 1 \\ \neg l_i, & \text{se } x_i = 0 \end{cases}$$

Tomemos o exemplo da Tabela 1. Às variáveis x_1, x_2 e x_3 fazemos corresponder os literais a, b e c, respetivamente. A linha da tabela de verdade para a qual $x_1 = 0$, $x_2 = 0$ e $x_3 = 0$ indica que f tem o valor 1 para esta combinação, portanto podemos extrair desta linha o conjuntório $(\neg a \wedge \neg b \wedge \neg c)$.

O disjuntório de todos os conjuntórios obtidos da tabela de verdade da função booleana f é uma fórmula de lógica proposicional na sua forma FND que é equivalente a f.

Portanto, por observação da Tabela 1, podemos obter a seguinte fórmula FND que é equivalente à função booleana f:

$$(\neg a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge b \wedge c)$$

É possível obter uma fórmula FNC equivalente a uma função booleana f calculando a negação da fórmula FND obtida a partir de $\neg f$ (obter uma fórmula FND a partir das linhas da tabela de verdade onde f tem valor 0 em vez de 1). Ou seja, se φ é uma fórmula FND equivalente a $\neg f$, $\neg\varphi$ é uma fórmula FNC equivalente a f.

Deste modo, da Tabela 1 obtemos a seguinte fórmula FND equivalente a $\neg f$

$$(\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge \neg c) \vee (a \wedge \neg b \wedge c)$$

e a sua negação

$$(a \vee b \vee \neg c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee c) \wedge (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c)$$

é uma fórmula FNC equivalente a f.

Template da solução a submeter

Deverá utilizar o template abaixo para escrever a solução deste problema.

```
(* Cabeçalho a incluir *)

type formula =
  | Lit of char
  | Neg of char
  | Conj of formula * formula
  | Disj of formula * formula

let rec compare_formula f_1 f_2 =
  match (f_2, f_1) with
  | Lit c_1, Lit c_2 | Neg c_1, Neg c_2 -> Char.compare c_1 c_2
  | Lit c_1, Neg c_2 when c_1 = c_2 -> -1
  | Neg c_1, Lit c_2 when c_1 = c_2 -> 1
  | (Lit c_1 | Neg c_1), (Lit c_2 | Neg c_2) -> Char.compare c_1 c_2
  | (Lit _ | Neg _), (Disj _ | Conj _) -> -1
  | (Disj _ | Conj _), (Lit _ | Neg _) -> 1
  | Conj (f_1_1, f_1_2), Conj (f_2_1, f_2_2)
```

```
  | Disj (f_1_1, f_1_2), Disj (f_2_1, f_2_2) ->
      let c = compare_formula f_1_1 f_2_1 in
      if c = 0 then compare_formula f_1_2 f_2_2 else c
  | Conj _, Disj _ | Disj _, Conj _ -> 0

let rec normalize_conjs acc f_1 f_2 =
  match (f_1, f_2) with
  | Conj (f_1_1, f_1_2), Conj (f_2_1, f_2_2) ->
      normalize_conjs (normalize_conjs acc f_1_1 f_1_2) f_2_1 f_2_2
  | (Lit _ | Neg _ | Disj _), Conj (f_1', f_2') ->
      normalize_conjs (normalize_formula f_1 :: acc) f_1' f_2'
  | Conj (f_1', f_2'), (Lit _ | Neg _ | Disj _) ->
      normalize_formula f_2 :: normalize_conjs acc f_1' f_2'
  | _ -> normalize_formula f_2 :: normalize_formula f_1 :: acc

and normalize_disjs acc f_1 f_2 =
  match (f_1, f_2) with
  | Disj (f_1_1, f_1_2), Disj (f_2_1, f_2_2) ->
      normalize_disjs (normalize_disjs acc f_1_1 f_1_2) f_2_1 f_2_2
  | (Lit _ | Neg _ | Conj _), Disj (f_1', f_2') ->
      normalize_disjs (normalize_formula f_1 :: acc) f_1' f_2'
  | Disj (f_1', f_2'), (Lit _ | Neg _ | Conj _) ->
      normalize_formula f_2 :: normalize_disjs acc f_1' f_2'
  | _ -> normalize_formula f_2 :: normalize_formula f_1 :: acc

and normalize_formula = function
  | (Lit _ | Neg _) as f -> f
  | Conj (f_1, f_2) -> (
      match normalize_conjs [] f_1 f_2 |> List.sort compare_formula
        with
      | x :: xs -> List.fold_left (fun f acc -> Conj (acc, f)) x xs
      | _ -> assert false)
  | Disj (f_1, f_2) -> (
      match normalize_disjs [] f_1 f_2 |> List.sort compare_formula
        with
      | x :: xs -> List.fold_left (fun f acc -> Disj (acc, f)) x xs
      | _ -> assert false)

exception Malformed

let normalize_disjs f =
  let rec aux acc = function
    | (Lit _ | Neg _ | Conj _) as f -> f :: acc
    | Disj (((Lit _ | Neg _ | Conj _) as f_1), f_2) -> aux (f_1 ::
        acc) f_2
```

```
    | Disj (Disj _, _) -> raise Malformed
  in
  aux [] f |> List.rev

let normalize_conjs f =
  let rec aux acc = function
    | (Lit _ | Neg _ | Disj _) as f -> f :: acc
    | Conj (((Lit _ | Neg _ | Disj _) as f_1), f_2) -> aux (f_1 ::
        acc) f_2
    | Conj (Conj _, _) -> raise Malformed
  in
  aux [] f |> List.rev

let string_of_formula =
  let rec aux conj disj f = function
    | Lit c -> f (Char.escaped c)
    | Neg c -> f ("!" ^ Char.escaped c)
    | Conj (f_1, f_2) ->
        aux true false
          (fun s_1 ->
            aux true false
              (fun s_2 ->
                f
                  (if conj then s_1 ^ " & " ^ s_2
                  else "(" ^ s_1 ^ " & " ^ s_2 ^ ")"))
              f_2)
          f_1
    | Disj (f_1, f_2) ->
        aux false true
          (fun s_1 ->
            aux false true
              (fun s_2 ->
                f
                  (if disj then s_1 ^ " | " ^ s_2
                  else "(" ^ s_1 ^ " | " ^ s_2 ^ ")"))
              f_2)
          f_1
  in
  aux false false (fun x -> x)

let print_formula f = normalize_formula f |> string_of_formula |>
      print_endline

(* Escreva a solução do problema a seguir *)
```

Descarregar template