UL245

SESSION 2004

Filière : 2ème concours – Concours F/S (Paris)

INFORMATIQUE

(Epreuve commune aux ENS Ulm et Lyon)

Durée : 3 heures

L'usage des calculatrices électroniques de poche à alimentation autonome, sans imprimante et sans document d'accompagnement est autorisé.
Cependant, une seule calculatrice à la fois est admise sur la table ou le poste de travail. Aucun échange n'est permis entre les candidats.

Tournez la page S.V.P.

I. Important

Il est recommandé de lire l'ensemble du sujet avant de commencer la rédaction.

Les algorithmes seront rédigés dans un langage informatique précis au choix du candidat parmi les langages couramment utilisés. Les programmes seront correctement indentés et judicieusement commentés. Quelques lignes griffonnées de code approximatif ne peuvent constituer une réponse acceptable. Les occasions sont nombreuses de rédiger des algorithmes et des programmes propres : constantes symboliques, passage des paramètres, bonne prise en compte des cas à problème, utilisation raisonnable de la mémoire et des ressources. Le candidat prendra soin d'éviter des constructions inutilement absconses.

II. Préambule

Ce sujet traite de descriptions de circuits numériques. Pour décrire un circuit numérique :

- On définit un ensemble de variables booléennes (par exemple `A`), de tableaux de variables booléennes (notés avec l'indice entre crochets, par ex. `R[1]`), de tableaux de tableaux (par ex. `W[1][2]`) et ainsi de suite.
- Une variable booléenne a pour seules et uniques valeurs `VRAI` et `FAUX`.
- Une formule booléenne s'écrit exclusivement avec les opérateur `ET` et `OU` à deux entrées, l'opérateur `NON` à une entrée, les constantes `VRAI` et `FAUX` et les parenthèses. Les règles de priorité n'étant pas claires, le candidat prendra soin d'utiliser suffisamment de parenthèses pour éviter toute ambiguïté.
- Une variable est soit une entrée, soit définie par une formule booléenne, notée avec le signe = où la variable définie apparaît seule et à gauche (par ex. `A = B OU (NON (C))`).
- Un circuit est décrit par un ensemble de définitions. Chaque variable qui n'est pas une entrée doit être définie exactement une et une seule fois. Certaines cellules des tableaux peuvent ne pas être définies si elles ne sont jamais utilisées par la suite.
- **Aucune structure de contrôle comme un test ou une boucle ne doit apparaître dans la description d'un circuit. Il n'y a pas de notion de variable locale ni de notion de fonction dans une description de circuit.**

Nous allons dans ce problème écrire des programmes qui impriment à l'écran la description de circuits. Les fonctions que nous décrirons utilisent comme paramètres le nom des variables d'entrée (par ex. `A`, `B` et `C`) et le nom des variables définies par la fonction (par ex. `D` et `E`). Les variables définies sont utilisées pour les résultats intermédiaires et pour les résultats de la fonction.

Par exemple la fonction `ET_OU_3` pourra être définie pour que l'appel

```
ET_OU_3 (« A », « B », « C », « D », « E »)
```

produise l'affichage des lignes suivantes à l'écran

```
D = A ET B ET C
E = A OU B OU C
```

Les paramètres « A », « B », « C », « D » et « E » sont passés entre guillemets parce qu'il ne s'agit pas de variables du programme mais de variables de la description du circuit. Les chaînes sont reproduites à l'identique sans être interprétées.

On pourra aussi définir des fonctions capables de travailler sur des tableaux en omettant les crochets correspondants et en précisant l'indice de début, l'indice de fin et éventuellement le pas d'incrément des indices. Ainsi, la fonction `ET_3_Tableau` pourra être définie et utilisée pour initialiser toutes les valeurs `S[i]` pour `i` variant de 1 à 5 avec un pas de 2 avec l'appel suivant :

```
ET_3_Tableau (« R », « W[1] », « W[2] », « S », 1, 5, 2)
```

pour produire

```
S[1] = R[1] ET W[1][1] ET W[2][1]
S[3] = R[3] ET W[1][3] ET W[2][3]
S[5] = R[5] ET W[1][5] ET W[2][5]
```

Là encore « R », « W[1] », « W[2] », « S » ne sont pas des variables du programme mais des variables de la description. Voilà trois exemples pour la fonction ET_3_Tableau et l'appel principal. Le premier est en C++.

```
#include <iostream>
#include <ostream>
void ET_3_Tableau (char* A, char* B, char* C, char* D,
                   int debut, int fin, int pas) {
  for (int i = debut; i <= fin; i += pas)
    std::cout << D << "[" << i << "]" <<
       " = "  << A << "[" << i << "]" <<
       " ET " << B << "[" << i << "]" <<
       " ET " << C << "[" << i << "]" << std::endl;
}
main () { ET_3_Tableau("R", "W[1]", "W[2]", "S", 1, 5, 2); }
```

Le deuxième exemple est en Java.

```
import java.io.*;
class Circuit {
  public static void main(String[] arg)
  { ET_3_Tableau("R", "W[1]", "W[2]", "S", 1, 5, 2); }
  public static void ET_3_Tableau (String A, String B, String C, String D,
                                   int debut, int fin, int pas) {
    for (int i = debut; i <= fin; i += pas)
      System.out.println(          D + "[" + i + "]" +
                         " = "  + A + "[" + i + "]" +
                         " ET " + B + "[" + i + "]" +
                         " ET " + C + "[" + i + "]");
  }
}
```

Et le troisième est en CAML.

```
let et_3_tableau a b c d debut fin pas =
  let i = ref(debut) in
  while (!i <= fin) do
    let ii = string_of_int !i in
    print_string (          d ^ "[" ^ ii ^ "]" ^
                  " = "  ^ a ^ "[" ^ ii ^ "]" ^
                  " ET " ^ b ^ "[" ^ ii ^ "]" ^
                  " ET " ^ c ^ "[" ^ ii ^ "]");
    print_newline();
    i := !i + pas;
  done
let main = et_3_tableau "R" "W[1]" "W[2]" "S" 1 5 2
```

Le candidat devra s'assurer que ses programmes génèrent bien des descriptions cohérentes. Il pourra à loisir définir des fonctions plus compliquées mêlant des variables et des tableaux pour générer automatiquement des descriptions cohérentes de circuits.

III. Premiers exemples

1. Une cellule OU Exclusif est une cellule à deux entrées booléennes A et B et une sortie booléenne D. D est à VRAI, si et seulement si une et une seule des entrées A ou B est à VRAI.

Ainsi, D est à FAUX, si les deux entrées A et B sont simultanément à FAUX ou simultanément à VRAI. Donner une formule booléenne définissant D en fonction de A et B. Donner le code d'une fonction XOR qui décrit cette cellule avec comme paramètres les noms des variables A, B et D.

2. On peut vouloir réaliser un OU Exclusif, cellule par cellule, de deux tableaux de variables booléennes. Donner une fonction XOR_Tableau qui répète le OU Exclusif sur les cellules spécifiées de deux tableaux R et S avec les résultats dans des cellules d'un tableau T. On peut ne pas vouloir traiter toutes les cellules des tableaux, pour cela, on utilisera des paramètres distincts pour les indices des débuts et pour les pas d'incrément dans les tableaux R, S et T. Par contre, le OU Exclusif concernant le même nombre de variables booléennes dans ces trois tableaux, cette dernière donnée pourra être transmise par un paramètre unique.

3. Une cellule multiplexeur est une cellule à trois entrées booléennes A, B et C et une sortie booléennes D. Si C est à VRAI, D a la valeur de A, la première entrée, sinon, D a la valeur de B, la seconde entrée. Donner une formule booléenne définissant D en fonction de A, B et C puis le code d'une fonction MUX qui décrit cette cellule.

4. On peut vouloir multiplexer deux parties de tableaux de variables booléennes. Donner le code d'une fonction MUX_Tableau qui décrit un multiplexeur de deux tableaux R et S dans un tableau T d'après une commande booléenne C. On peut ne pas vouloir traiter toutes les cellules des tableaux, pour cela, on utilisera des paramètres distincts pour les indices des débuts et pour les pas d'incrément dans les tableaux R, S et T. Par contre le multiplexeur concernant le même nombre de variables booléennes dans ces trois tableaux, cette dernière donnée pourra être transmise par un paramètre unique.

IV. Compteur à codage 1 parmi N

Dans cette partie, nous allons décrire un circuit qui compte le nombre de variables à VRAI dans un tableau R de N-1 variables booléennes et positionne ensuite un et un seul signal à VRAI dans le tableau de sortie S de N variables tel que S[K] est à VRAI si et seulement si **exactement** K variables sont à VRAI dans le tableau R. On parle ici de codage **1 parmi N** car un et un seul signal est actif dans S.

5. Donner une fonction THERMOMETRE capable de décrire un circuit qui ne contient que les opérateurs ET et OU et qui positionne T[K] à VRAI si et seulement si **au moins** K variables sont à VRAI parmi les N-1 variables de R. On parle ici de codage **thermométrique**.

6. En utilisant la fonction XOR_Tableau définie précédemment, donner le code d'une première fonction PREMIER_THERMO_EN_1_PARMI_N capable de décrire un circuit transformant un codage thermométrique en codage 1 parmi N. Compter le nombre d'opérateurs NON apparaissant dans la description du circuit en fonction du paramètre N.

7. Donner le code d'une fonction THERMO_EN_1_PARMI_2 qui permet de transformer un codage thermométrique à deux signaux en un codage 1 parmi 2 et qui utilise autant d'opérateurs logiques ET et OU que nécessaire, mais une seule négation logique NON.

8. Proposer une méthode pour transformer un codage thermométrique à 2N entrées en un codage 1 parmi 2N en utilisant THERMO_EN_1_PARMI_N pour un code à N entrées, une cellule THERMO_EN_1_PARMI_2 et uniquement des opérateurs logiques ET et OU. *Indications* :

   - Coupler les variables consécutives du tableau d'entrée deux par deux pour former un tableau intermédiaire de N entrées.

- Introduire deux variables intermédiaires F et G telles que F est VRAI si la première variable du couple désigné par la transformation I parmi N est VRAI et G est VRAI si la seconde variable du couple désigné par la transformation I parmi N est VRAI.

9. Donner une fonction récursive THERMO_EN_1_PARMI_N qui permet de transformer un codage thermométrique à N signaux en un codage 1 parmi N quand N est une puissance de 2. Calculer le nombre total d'opérateurs NON utilisés. On pourra utiliser un tableau de tableaux W pour définir les variables intermédiaires. Cette fonction sera utilisée pour la question **14.**

V. Négations logiques avec moins d'opérateurs NON

Dans cette partie, nous allons décrire un circuit qui fait en sorte que chaque cellule du tableau de sortie U est la négation logique de la cellule correspondante du tableau d'entrée V. On donne les indices de début, de fin et le pas d'incrément dans les tableaux. Ainsi,

```
INV (V, U, 1, 3, 1)
```

pourrait produire le résultat suivant :

```
U[1] = NON (V[1])
U[2] = NON (V[2])
U[3] = NON (V[3])
```

Nous allons décrire un circuit alternatif équivalent qui utilise **moins de 3 opérateurs de négation logique**.

10. Vérifier que si toutes les variables de V sont à FAUX, on obtient le résultat attendu en définissant toutes les valeurs de U à VRAI. Écrire le code d'une fonction INV_N_SACHANT_0_VRAI qui décrit le circuit correspondant.

11. Vérifier que si l'on sait qu'une et une seule des variables de V est à VRAI on peut définir correctement toutes les variables de U en n'utilisant que l'opérateur OU et aucune négation logique. Écrire le code d'une fonction INV_N_SACHANT_1_VRAI qui décrit le circuit correspondant.

12. Vérifier que si l'on sait qu'exactement K variables parmi les N de V sont à VRAI on peut définir toutes les variables de U en n'utilisant que les opérateurs ET et OU et aucune négation logique. Écrire le code d'une fonction INV_N_SACHANT_K_VRAI qui décrit le circuit correspondant.

13. Calculer le nombre d'opérateurs ET et OU apparaissant dans la description du circuit générée par l'opérateur INV_N_SACHANT_K_VRAI en fonction des paramètres N et K.

14. En utilisant les fonctions INV_N_SACHANT_K_VRAI, THERMOMETRE et THERMO_EN_1_PARMI_N, proposer une fonction INV qui décrit un circuit qui calcule la négation logique de chacune des N cellules du tableau en entrée. Donner en fonction de N le nombre de fois où apparaissent chacun des opérateurs logiques ET, OU et NON dans la description.

15. Conclure dans le cas où N vaut 3.

16. *Question subsidiaire et ouverte* : Peut-on proposer une méthode utilisant encore moins d'opérateurs NON, dans le cas où N=3, dans le cas général ?