Centrale Informatique MP 2006 — Corrigé

Ce corrigé est proposé par Samuel Mimram (ENS Lyon) ; il a été relu par Sattisvar Tandabany (ENS Lyon) et Vincent Puyhaubert (Professeur en CPGE).

Le sujet est constitué de deux parties indépendantes.

- La première propose d'implémenter un algorithme de compression de données en regroupant dans des documents des occurrences consécutives d'un même caractère. L'écriture de fonctions (courtes) est demandée.
- La seconde partie, plus longue, introduit des problèmes de logique qu'un joueur, enfermé dans un labyrinthe, doit résoudre pour pouvoir positionner des leviers qui vont lui permettre de sortir du labyrinthe. Trois techniques principales sont abordées. Il est tout d'abord proposé de résoudre les énigmes en les formalisant sous forme de formules en logique booléenne. Les tables de vérité et les identités classiques sur les formules en logique booléenne (lois de De Morgan, etc.) sont mises à contribution. On est ensuite amené à réaliser le circuit électrique du labyrinthe à l'aide de portes logiques. Enfin, il est proposé de réaliser un automate afin de vérifier les solutions du joueur au problème logique.

L'énoncé comporte principalement des questions pratiques et permet donc de réviser la manipulation des outils classiques.

Indications

Partie I

I.A.1 Utiliser l'encodeur pour stocker le nombre d'occurrences consécutives déjà lues du dernier caractère rencontré.

I.A.2 Mettre à jour l'encodeur en testant si le caractère en entrée est le même que le dernier caractère lu.

I.A.3 S'intéresser à la longueur du codage.

I.A.4 Ne pas oublier que n est stocké sous la forme de son écriture décimale.

Partie II

II.D Dresser les tables de vérité des formules E_1, E_2 et E_3.

II.E Dresser la table des valeurs possibles pour L_4, L_5 et L_6 pour chacune des trois parties de l'énoncé.

II.F Idem.

II.G Donner des formules simples qui sont vraies si et seulement si les portes 1 et 2 doivent s'ouvrir puis les implémenter en utilisant les portes logiques.

II.H Proposer un circuit contenant une boucle de rétroaction. Cette question est difficile et hors-programme, n'hésitez pas à passer directement à la question II.J.

II.I Utiliser le circuit mémoire de la question précédente pour stocker la validité des positions des trois premiers leviers.

II.J En utilisant les valeurs précédemment trouvées pour L_1, …, L_6 énumérer les valeurs des nombres ayant pour écriture binaire les leviers, suivant les différentes positions possibles des leviers L_7, L_8 et L_9.

II.K Utiliser la relation $\overline{b_1 \cdots b_{k+1}} = 2 \times \overline{b_1 \cdots b_k} + \overline{b_{k+1}}$.

II.L Appliquer le résultat de la question précédente.

I. Un algorithme de compression de données

> Les identifiants en OCaml ne prennent pas de majuscules (sauf les noms de modules et les constructeurs de types inductifs). Nous noterons donc `symbole_final` le symbole qui termine un document et `encodeur` le type des encodeurs afin que le code présenté soit valide à la fois en Caml Light et en OCaml.

I.A.1 Nous allons utiliser les encodeurs (les enregistrements de type `encodeur`) de la façon suivante. Le champ `code_symbole` contiendra le code du dernier symbole lu et le champ `compte` contiendra le nombre d'occurrences consécutives déjà lues de ce symbole.

La fonction `initialiser_coder` crée un nouvel encodeur. Son champ `compte` contiendra 0 – aucun caractère n'a encore été lu – et un symbole quelconque dans `code_symbole` – nous avons ici choisi `symbole_final`.

```
let initialiser_codeur () =
  {
    code_symbole = symbole_final;
    compte = 0
  }
;;
```

La fonction `vider_codeur` doit placer sur la sortie le symbole dont le code est contenu dans le champ `code_symbole`, éventuellement précédé de `[`n`]` si le nombre n d'occurrences de ce symbole (stocké dans le champ `compte` de l'encodeur) est strictement supérieur à 1. Si n est nul, il ne faut bien sûr rien placer du tout sur la sortie. Une fois cette opération réalisée, il faut remettre le compteur d'occurrences de l'encodeur à 0. En voici une implémentation possible :

```
let vider_codeur encodeur =
  if encodeur.compte > 1 then
    (
      sortir_char '[';
      sortir_int encodeur.compte;
      sortir_char ']';
    );
  if encodeur.compte > 0 then
    sortir_code encodeur.code_symbole;
  encodeur.compte <- 0;
  encodeur
;;
```

I.A.2 La fonction `coder` doit incrémenter le compteur d'occurrences (le champ `compte`) de l'encodeur si le symbole donné en argument est le même que celui qui a été précédemment lu et qui est stocké dans le champ `code_symbole` de l'encodeur. Sinon, il faut placer sur la sortie le contenu de l'encodeur (grâce à la fonction `vider_codeur`), puis stocker le symbole donné en argument en tant que symbole précédemment lu et remettre le compteur d'occurrences à 1.

```
let coder encodeur symb =
  if symb = encodeur.code_symbole then
    encodeur.compte <- encodeur.compte + 1
  else
    (
      vider_codeur encodeur;
      encodeur.code_symbole <- symb;
      encodeur.compte <- 1;
    );
  encodeur
;;
```

> Le sujet ne précise pas ce qu'il faut faire si le symbole donné en argument est `symbole_final`. Nous n'avons pas géré ce cas ici, mais il était par exemple envisageable de lever une exception.
>
> Le sujet précise que les fonctions `vider_codeur` et `coder` doivent renvoyer les nouveaux états des encodeurs. Ceci est inutile car les champs des encodeurs sont modifiables (ce qui est déclaré par le mot clé `mutable`).

I.A.3 Le résultat du codage du document `aabbcc` avec ce algorithme est

`[2]a'[2]b'[2]c'.`

On constate qu'il est plus long (en nombre de caractères) que l'original ! Pour remédier à ce problème, il suffit de ne mettre une séquence c^n, contenant n caractères consécutifs c, sous la forme `[`n`]`c que si n est supérieur à 4. En effet, en deçà le codage produit une séquence plus longue que l'originale.

> Pour tester nos fonctions, supposons que le code d'un caractère est son code ASCII (l'un des systèmes de codage les plus utilisés). On peut définir les fonctions de l'énoncé de la façon suivante :
>
> ```
> let symbole_final = 0;;
>
> let code = int_of_char;;
>
> let symbole = char_of_int;;
>
> let sortir_char = print_char;;
>
> let sortir_code c = sortir_char (symbole c);;
>
> let sortir_int = print_int;;
> ```
>
> Par convention, nous avons supposé que le code du symbole final est 0, ce qui correspond au caractère spécial `'\000'`. La fonction `symbole_suivant` peut être simulée par la fonction suivante. Elle renvoie un à un les caractères de la chaîne de caractères `document_test` qui contient l'exemple `aabbcc` de la question I.A.3 et se termine par le caractère `'\000'`.
>
> ```
> let document_test = "aabbcc\000";;
>
> let i = ref (-1);;
> ```