Le problème impossible

♢
Un
mathématicien
coquin choisit
un jour deux
nombres entiers (x
et y bien sûr, avec $x \leq y$*)*
dans l'intervalle $[2, 100]$*, et en*
fit la somme (qu'il nota s : on peut
être coquin et manquer d'imagination)
et le produit (noté désormais p). Il appela
alors l'un de ses coreligionnaires M_1 *et lui*
indiqua la somme s en précisant que $x, y \in [2, 100]$*.*
Ne regardant pas à la dépense, il expédia alors un
télégramme au célèbre mathématicien M_2 *pour lui indiquer*
la valeur de p ainsi que le renseignement sur x et y. Le but de
M_1 *et* M_2 *était alors de trouver x et y. Evidemment* M_1 *n'avait*
pas le droit de communiquer s à M_2 *et ce dernier n'aurait*
su dévoiler p à M_1 *(et l'on sait que les mathématiciens*
sont des gens fort honnêtes...). Rapidement, M_1
joignit M_2 *et lui dit «* tu ne peux pas résoudre
le problème ! *». Quelque temps plus*
tard, M_2 *appella* M_1 *et lui déclara :*
« j'ai résolu le problème ! *».*
M_1*, un peu dépité de s'être*
fait griller, annonca
alors peu après :
« finalement,
j'ai moi
aussi
vaincu ! *».*
♢

Quels sont les entiers x et y ?

Ce problème, ainsi posé par Martin Gardner dans la revue *Scientific American* paraît impossible à résoudre (d'où le titre...) : il semble qu'il manque des informations. Cependant, tout cela n'est qu'apparence et il est possible de déterminer de manière certaine les entiers x et y. Précisons d'entrée que M_1 et M_2 savent tous les deux que x et y sont des entiers compris entre 2 et 100 et savent également ce qui a été communiqué à leur collègue. Cela va être capital !

1 Le crible d'Ératosthène

Soit n un entier fixé. Le crible d'Ératosthène est un algorithme qui permet de déterminer de manière efficace les entiers premiers strictement inférieurs à n. On construit pour cela un tableau t de booléens de taille n dont toutes les cases initialisées à true, sauf les cases 0 et 1 qui contiennent initialement false. Pour i variant de 2 à la partie entière de $\sqrt{n}$, on itère alors le procédé suivant : si $t.(i)$ contient true alors on met false dans toutes les cases du tableau dont l'indice est un multiple *strict* de i. À la fin de l'algorithme, $t.(i)$ contient true si et seulement si l'entier i est premier.

▶ **Question 1** *Écrivez une fonction* eratosthene *qui implémente cet algorithme.*

value eratosthene : *int* → *bool vect*

▶ **Question 2** *Écrivez une fonction* decompose_somme *telle que* decompose_somme t s *retourne un booléen indiquant si l'entier s se décompose comme la somme de deux entiers premiers. (L'argument supplémentaire t est un tableau de booléens de taille supérieure à s tel que $t.(x)$ indique si x est premier.)*

value decompose_somme : *bool vect* → *int* → *bool*

2 Une première analyse

On peut commencer par interpréter le premier appel de M_1 : s'il affirme que M_2 ne peut pas résoudre le problème, c'est qu'en aucun cas s ne se décompose comme la somme de deux nombres premiers ; sinon il y aurait une chance que M_2 puisse conclure.

▶ **Question 3** *Écrivez une fonction* sommes *prenant un entier n pour argument et retournant la liste des sommes envisageables à cet instant si $x, y \in [\![2, n]\!]$.*

value sommes : *int* → *int list*

On peut en fait éliminer beaucoup plus de sommes. Supposons, avec les données du problème, que $s = x + 53$ avec $x \in [\![2, 100]\!]$. Il serait alors possible que $p = x \times 53$, or ce dernier nombre se décompose de façon unique comme un produit de deux entiers *compris entre 2 et 100*. Dans cette situation, M_1 ne pourrait affirmer que M_2 ne peut pas résoudre le problème ! Ainsi, toutes les sommes de ce type sont impossibles, c'est à dire toutes les sommes de la forme $s = x + y$ où $x \in [\![2, n]\!]$ et y est un entier premier supérieur à $n/2$.

▶ **Question 4** *Reprenez la fonction* sommes *en appliquant ce qui vient d'être expliqué et appliquez cette fonction au cas $n = 100$.*

value sommes' : *int* → *int list*

Vous devez normalement trouver les onze sommes de l'ensemble e_s suivant :

$$e_s = \{11; 17; 23; 27; 29; 35; 37; 41; 47; 51; 53\}$$

On peut vérifier que cet ensemble est « minimal » : si $s = x + y \in e_s$ alors le nombre $p = x \times y$ peut effectivement se décomposer de plusieurs façons.

3 Produits plausibles

M_2 sait maintenant que $s \in e_s$. Illustrons sa réflexion par un exemple. Supposons que $p = 130$. M_2 peut alors envisager trois décompositions : 2×65, 5×26 et 10×13. Dans le premier cas, la somme vaudrait 67, dans le second 31 et 23 dans le troisième. Parmi ces trois nombres, seul 23 est élément de e_s et donc M_2 peut affirmer que $x = 10$ et $y = 13$. Nous dirons que cette valeur de p est plausible.
À l'inverse, la valeur $p = 132$ n'est pas plausible : $132 = 3 \times 44 = 11 \times 12$ avec $11 + 12 = 23$ et $44 + 3 = 47$ tous deux dans e_s. M_2 ne pourrait donc pas logiquement rappeler M_1 dans ce cas.

▶ **Question 5** *Écrivez une fonction* decompose_produit *qui prend pour argument un entier p et qui retourne la liste des couples (x, y) d'entiers supérieurs ou égaux à 2 tels que $x \leq y$ et $p = x \times y$.*

value decompose_produit : *int* → *int* × *int list*

▶ **Question 6** *Déduisez-en une fonction* plausible *qui prenne en arguments p ainsi que e_s et renvie* true *si p est plausible et* false *sinon.*

value plausible : *int list* → *int* → *bool*

▶ **Question 7** *Calculez ensuite l'ensemble e_p des valeurs plausibles de p (vous devez en trouver 102).*

4 Sommes miraculeuses

Il reste à utiliser le dernier appel de M_1. Reprenons pour cela notre exemple précédent en montrant que la valeur plausible $p = 130$ n'est pas valable.
On suppose donc que $p = 130$ et $s = 23$. M_1 sait que le couple (x, y) est dans l'ensemble $\{(2, 21); (3, 20); \ldots; (11, 12)\}$. Il peut donc envisager dix valeurs pour le produit : $42, 60, \ldots, 132$. Parmi ces valeurs se trouvent 130 ($= 10 \times 14$) et 112 ($= 7 \times 16$). Or 112 et 130 sont tous deux plausibles et donc, si s valait 23, M_1 n'aurait pas pu, à son tour, deviner x et y !
Pour que les choses se soient passées comme elles se sont passées, il faut que, parmi toutes les décompositions de $s = u + v$, il n'y en ait qu'une seule telle que $u \times v$ soit un produit plausible ! Une telle valeur de s sera dite miraculeuse.

▶ **Question 8** *Écrivez une fonction* miraculeuse *qui prenne s et e_p comme arguments et renvoie* true *si s est miraculeuse et* false *sinon.*

value miraculeuse : *int list* → *int* → *bool*

▶ **Question 9** *Vérifiez qu'il n'y a ici qu'une seule valeur miraculeuse. Déterminez les valeurs de x et y.*

▶ **MPSI – Option Informatique**
Année 2002, Dixième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Le problème impossible

Un corrigé

▶ Question 1

```
let eratosthene n =
  let t = make_vect n true in
  t.(0) <- false;
  t.(1) <- false;
  for i = 2 to n - 1 do
    if t.(i) then begin
      let j = ref (2 * i) in
      while !j < n do
        t.(!j) <- false;
        j := !j + i
      done
    end
  done;
  t
;;
```

▶ Question 2

```
let decompose_somme t s =
  let res = ref false in
  for x = 2 to s/2 do
    if t.(x) && t.(s - x) then res := true
  done;
  ! res
;;
```

▶ Question 3

```
let rec sommes n =
  let t = eratosthene (2 * n + 1) in
  let rec aux = function
      3 -> []
    | s ->
        if decompose_somme t s
        then aux (s - 1)
        else s :: aux (s - 1)
  in
  aux (2 * n)
;;
```

▶ Question 4

```
let decompose_somme' t n s =
  let res = ref false in
  for x = 2 to s/2 do
    if (t.(x) or (s - x) >= n / 2)
    && t.(s - x) then res := true
  done;
  ! res
;;

let rec sommes' n =
  let t = eratosthene (2 * n + 1) in
  let rec aux = function
      3 -> []
    | s ->
        if decompose_somme' t n s then
        then aux (s - 1)
        else s :: aux (s - 1)
  in
  aux (2 * n)
;;
```

▶ Question 5

```
let decompose_produit p =
  let rec aux = function
      1 -> []
    | x ->
      if p mod x > 0 or x >= p / x
      then aux (x - 1)
      else (x, p / x) :: aux (x - 1)
  in
  aux (p / 2)
;;
```

▶ Question 6

```
let plausible e_s p =
  let rec compte = function
      [] -> 0
    | (x, y) :: q ->
      if mem (x + y) e_s
      then 1 + compte q
      else compte q
  in
  compte (decompose_produit p) = 1
;;
```

▶ Question 7

```
let rec plausibles e_s = function
    3 -> []
  | p ->
    if plausible e_s p
    then p :: plausibles e_s (p - 1)
    else plausibles e_s (p - 1)
;;

let e_p = plausibles e_s (53 * 53)
;;
```

▶ Question 8

```
let miraculeuse e_p s =
  let i = ref 0 in
  for x = 2 to s / 2 do
    if mem (x * (s - x)) e_p
    then i := !i + 1
  done;
  !i = 1
;;
```

▶ Question 9

```
let miraculeuses e_p e_s =
  let rec filtre = function
      [] -> []
    | s :: q ->
        if miraculeuse e_p s
        then s :: filtre q
        else filtre q
  in
  filtre e_s
;;

let solution e_p s =
  let x = ref 2 in
  while not (mem (!x * (s - !x)) e_p) do
    x := !x + 1
  done;
  (!x, s - !x)
;;
```

```
#solution e_p 17;;
- : int * int = 4, 13
```