Programmes Corrects par Construction
Partie 2 : Exemples - Plus long segment avec au plus K valeurs nulles

Pierre-Edouard Portier

1 Plus long segment avec au plus K valeurs nulles

```
  var f(0..N-1): array of int {N≥0}
; var K: int {K≥0}
; var r: int
r: R

R: r = (↑p,q : 0≤p≤q≤N ∧ Z.p.q ≤ K : q-p)

Z.p.q ≜ (#i : p≤i<q : f.i = 0)
```

Pour un segment vide, `Z.p.q` = `0` ≤ `K`.

Introduction d'un invariant par la stratégie de remplacement de la constante `N` par une variable `n`.

```
P0: 0≤n≤N
P1: r = (↑p,q : 0≤p≤q≤n ∧ Z.p.q ≤ K : q-p)
```

D'où une première esquisse de programme.

```
  n,r := 0,0
; do n≠N ->
    {? P1[n\n+1]}
  ; n := n+1
  od
```

Calcul de `P1[n\n+1]` pour construire la mise à jour de `r` qui permet le maintien de l'invariant à chaque itération de la répétition.

```
  P1[n\n+1]
=
  r' = (↑p,q : 0≤p≤q≤n+1 ∧ Z.p.q ≤ K : q-p)
= {split pour q=n+1, n<N, P1}
  r' = r ↑ (↑p : 0≤p≤n+1 ∧ Z.p.(n+1) ≤ K : n+1-p)
= {dualité ↓/↑, distributivité de + sur ↑}
  r' = r ↑ n+1 - (↓p : 0≤p≤n+1 ∧ Z.p.(n+1) ≤ K : p)
```

Introduction d'une nouvelle variable contrôlée par un nouvel invariant.

```
P2: s = (↓p : 0≤p≤n ∧ Z.p.n ≤ K : p)
```

D'où une nouvelle esquisse de programme.

```
  n,r := 0,0
; do n≠N ->
    {? P2[n\n+1]}
  ; r := r ↑ (n+1-s)
  ; n := n+1
  od
```

Calcul de `P2[n\n+1]` pour construire la mise à jour de `s` qui permet le maintien de l'invariant à chaque itération de la répétition.

```
  P2[n\n+1]
=
  s' = (↓p : 0≤p≤n+1 ∧ Z.p.(n+1) ≤ K : p)
= {split pour p=n+1, n<N, Z.(n+1).(n+1)=0, K≥0}
  s' = (↓p : 0≤p≤n ∧ Z.p.(n+1) ≤ K : p) ↓ (n+1)
```

Pour exprimer la mise à jour de `s` en fonction de son ancienne valeur, il faut faire apparaitre `Z.p.n ≤ K` au lieu de `Z.p.(n+1) ≤ K`.

```
  Z.p.(n+1) ≤ K
=
  (#i : p≤i<n+1 : f.i = 0) ≤ K
= {split pour i=n, p≤n}
  Z.p.n + #.(f.n=0) ≤ K
= {alternative, soit f.n=0 soit f.n≠0}
  if f.n≠0 -> Z.p.n ≤ K
    |f.n=0 -> Z.p.n ≤ K-1
  fi
```

Ainsi, pour mettre à jour `s` afin de maintenir `P2`, il faut, lorsque `f.n=0`, connaitre la valeur de :
`(↓p : 0≤p≤n ∧ Z.p.n ≤ K-1 : p)`
Pour cela, il faut introduire une nouvelle variable avec son invariant. Pour maintenir cet invariant, il faut, lorsque `f.n=0`, connaitre la valeur de :
`(↓p : 0≤p≤n ∧ Z.p.n ≤ K-2 : p)`
Etc.
Au lieu de la variable `s`, il faut introduire un tableau et son invariant.

```
var s(0..K): array of int

P2: (∀ k : 0≤k≤K : s.k = (↓p : 0≤p≤n ∧ Z.p.n ≤ k : p))
```

Pour calculer la mise à jour des éléments du tableau `s` qui permet de maintenir l'invariant, il faut distinguer deux cas : `0<k≤K` et `k=0`.
Pour `0<k≤K` :

```
  s.k' = (↓p : 0≤p≤n+1 ∧ Z.p.(n+1) ≤ k : p)
= {split pour p=n+1, n<N, Z.(n+1).(n+1)=0, k≥0}
  s.k' = (↓p : 0≤p≤n ∧ Z.p.(n+1) ≤ k : p) ↓ (n+1)
= {Z.n.(n+1) ≤ 1 ≤ k}
  s.k' = (↓p : 0≤p≤n ∧ Z.p.(n+1) ≤ k : p)
= {alternative, soit f.n=0 soit f.n≠0, voir supra}
  if f.n≠0 -> s.k' = s.k
    |f.n=0 -> s.k' = s.(k-1)
  fi
```

Pour `k=0` :

```
  s.0' = (↓p : 0≤p≤n+1 ∧ Z.p.(n+1) ≤ 0 : p)
= {split pour p=n+1, n<N, Z.(n+1).(n+1)=0}
  s.0' = (↓p : 0≤p≤n ∧ Z.p.(n+1) ≤ 0 : p) ↓ (n+1)
= {alternative, soit f.n=0 soit f.n≠0, voir supra
   avec Z.p.n≤-1 ⇒ ⊥}
  if f.n≠0 -> s.0' = s.0
    |f.n=0 -> s.0' = n+1
  fi
```

D'où le programme suivant. La notation `|[]|` indique un bloc de code, avec un sens similaire aux accolades `{}` du langage C.

```
  var f(0..N-1): array of int {N≥0}
; var s(0..K): array of int {K≥0}
; var n, r: int
; n,r := 0,0
; |[ var a: int ; a := 0 ; do a≠K+1 -> s.a := 0 ; a := a+1 od ]|
{P0, P1, P2}
; do n≠N ->
    if f.n≠0 -> skip
      |f.n=0 -> |[ var a: int
                 ; a := k
                 ; do a≠0 -> s.a := s.(a-1) ; a := a+1 od
                 ; s.0 := n+1
                |]
    fi
  ; r := r ↑ (n+1-s.K)
  ; n := n+1
  od
```

Dans le pire des cas, l'exécution de cet algorithme nécessite `K+N*K` étapes. Dans la boucle intérieure, le tableau `s` subit une rotation de 1 élément. En représentant `s` comme une liste circulaire, la rotation correspond à un déplacement de l'origine. En notant $\oplus$ l'addition modulo `K+1`, l'invariant `P2` devient :

```
P2: (∀ k : 0≤k≤K : s.(h⊕k) = (↓p : 0≤p≤n ∧ Z.p.n ≤ k : p))
```

D'où le programme suivant de complexité $O(N + K)$.

```
  var f(0..N-1): array of int {N≥0}
; var s(0..K): array of int {K≥0}
; var n,r,h: int
; n,r,h := 0,0,0
; |[ var a: int ; a := 0 ; do a≠K+1 -> s.a := 0 ; a := a+1 od ]|
{P0, P1, P2}
; do n≠N ->
    if f.n≠0 -> skip
      |f.n=0 -> h := h⊕K ; s.h := n+1
    fi
  ; r := r ↑ (n+1-s.(h⊕K))
  ; n := n+1
  od
```