Programmes Corrects par Construction
Exemples - Plus longue sous-séquence croissante

Pierre-Edouard Portier

```
var f(0..N-1): array of int {N>0}
```

Une sous-séquence de taille `s`, $0 \leq s \leq N$, est obtenue en retirant `N-s` éléments de `f` tout en conservant pour les `s` éléments leur ordre relatif dans `f`. Ainsi, `f` possède 2^N sous-séquences.

Il s'agit de construire un programme pour atteindre la post-condition suivante.

```
R: k = longueur maximale d'une sous-séquence croissante (ssc) de f
```

Puisque chaque élément de `f` doit être lu au moins une fois, un invariant est proposé en remplaçant la constante `N` par une variable `n` dans `R`.

```
P0: 1≤n≤N
P1: k = longueur maximale d'une ssc de f(i: 0≤i<n)
```

D'où l'esquisse de programme suivante :

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

Dans le corps de la boucle, il s'agit de déterminer si `k` doit ou non être incrémentée.

```
  {P1 ∧ 1≤n<N}
  if ... -> k:= k+1
   | ... -> skip
  fi
  {P1[n\n+1]}
; n:= n+1
```

L'alternative dépend de la découverte de la nouvelle valeur `f.n`.

```
  {P1 ∧ 1≤n<N}
  if m≤f.n -> k:= k+1
   | m>f.n -> skip
  fi
  {P1[n\n+1]}
; n:= n+1
```

L'introduction de la variable `m` doit être contrôlée par un nouvel invariant.

```
P2: m = le plus petit parmi les éléments en dernière position
        d'une ssc de longueur k dans f(i: 0≤i<n)
```

```
  n,k,m := 1,1,f.0 {P0 ∧ P1 ∧ P2}
; do n≠N ->
    {P1 ∧ P2 ∧ 1≤n<N}
    if m≤f.n -> k:= k+1 ; m:= f.n
     | m>f.n -> ...
    fi
  ; n:= n+1
  od
```

Dans la seconde branche de l'alternative, la connaissance de `f.n` permet-elle de diminuer `m` ?

```
  {f.n<m}
  if m'≤f.n -> m:= f.n
   | f.n<m' -> skip
  fi
```

L'introduction de la variable `m'` doit être contrôlée par un nouvel invariant.

```
P2': m' = le plus petit parmi les éléments en dernière position
          d'une ssc de longueur k-1 dans f(i: 0≤i<n)
```

Pour maintenir cet invariant, une variable `m''` est nécessaire, etc.
Ainsi, il faut introduire un tableau `m(j: 1≤j≤k)` avec l'invariant :

```
P2: (∀j : 1≤j≤k : m.j = le plus petit parmi
      les éléments en dernière position d'une ssc
      de longueur j dans f(i: 0≤i<n) )
```

L'ancien `m` devient `m.k`, `m'` devient `m.(k-1)`, etc.

```
  n,k,m.1 := 1,1,f.0 {P0 ∧ P1 ∧ P2}
; do n≠N ->
    if m.k≤f.n      -> k:= k+1 ; m.k:= f.n
     | m.1>f.n      -> m.1:= f.n
     | m.1≤f.n<m.k  -> "trouver j tel que m.(j-1)≤f.n<m.j "
                       ; m.j:= f.n
    fi
  ; n:= n+1
  od
```

Un dernier invariant, P3, contrôle la recherche binaire utilisée pour trouver j :

```
P3: m.i≤f.n<m.j ∧ 1≤i<j≤k
```

```
  {m.1≤f.n<m.k}
  i,j:= 1, k {P3}
; do i≠(j-1) ->
    h:= (i+j) div 2 {i<h<j}
  ; if m.h≤f.n -> i:= h {P3}
     | f.n<m.h -> j:= h {P3}
    fi {P3}
  od {m.(j-1)≤f.n<m.j}
```