Programmes Corrects par Construction
Partie 2 : Exemples - Segment AB Maximal

Pierre-Edouard Portier

1 Segment AB maximal

Parmi les segments du tableau `f` dont la première valeur est `A`, dont la dernière valeur est `B`, trouver l'un de ceux dont la somme des éléments est maximale. Il faut d'abord formaliser cette spécification :

```
R: r = (↑i,j : 0≤i<j<N ∧ f.i=A ∧ f.j=B : S.i.j)

avec S.i.j = (Σk : i≤k≤j : f.k)

  var A,B : int
; var f(0..N-1) : array of int {1≤N}
; var r : int
r: R
```

En prévision d'un parcours linéaire du tableau, un invariant est introduit par la transformation de la constante `N` en une variable `n`.

```
P0: 1≤n≤N
P1: r = (↑i,j : 0≤i<j<n ∧ f.i=A ∧ f.j=B : S.i.j)
```

D'où l'esquisse de programme suivante :

```
  n:=1 ; r:=-∞
; {inv P0 ∧ P1} {FdP N-n}
  do n≠N ->
    r: S
  ; {?P1[n\n+1]}
    n:=n+1
  od
  {R}
```

L'instruction `n:=n+1` assure l'évolution de la fonction de progrès par un parcours linéaire du tableau. Par définition de l'affectation, pour que l'invariant soit maintenu, la précondition de l'instruction `n:=n+1` doit être au moins aussi forte que l'assertion `P1[n\n+1]`. Ainsi, il faut construire un programme `S` qui réalise cette assertion. `S` doit mettre à jour la valeur de `r` pour que `P1[n\n+1]` soit vérifiée.

```
Re P1[n\n+1]

  (↑i,j : 0≤i<j<n+1 ∧ f.i=A ∧ f.j=B : S.i.j)
= { split pour j=n, correct car n≥1 par P0,
    P1 }
  r ↑ (↑i : 0≤i<n ∧ f.i=A ∧ f.n=B : S.i.n)
= { soit f.n=B, soit f.n≠B, correct car 0≤n<N,
    le max sur un domaine vide vaut -∞}
  if f.n≠B -> r
    |f.n=B -> r ↑ (↑i : 0≤i<n ∧ f.i=A : S.i.n)
  fi
```

La valeur : `(↑i : 0≤i<n ∧ f.i=A : S.i.n)` peut-elle être calculée avant la mise à jour de la variable `r` ? Une nouvelle variable, `s`, est introduite pour stocker cette valeur. Puisque la valeur de `s` est utilisée pour le calcul de la mise à jour de `r`, `s` doit être mise à jour avant `r`. L'introduction d'une nouvelle variable s'accompagne d'un renforcement de l'invariant.

```
P2: s=(↑i : 0≤i<n-1 ∧ f.i=A : S.i.(n-1))

  n:=1 ; r:=-∞ ; s:=-∞
; {inv P0 ∧ P1 ∧ P2} {FdP N-n}
  do n≠N ->
    s: T
  ; {?P2[n\n+1]}{P0}{P1}
    if f.n≠B -> skip
      |f.n=B -> r:= r↑s
    fi
  ; {P1[n\n+1]}
    n:=n+1
  od
  {R}
```

Il faut réaliser `P2[n\n+1]` pour construire le programme `T` qui met à jour la variable `s`.

```
Re P2[n\n+1]

    (↑i : 0≤i<n ∧ f.i=A : S.i.n)
=  { S.i.n = f.n + S.i.(n-1),
     + distribue sur ↑ }
   f.n + (↑i : 0≤i<n ∧ f.i=A : S.i.(n-1))
=  { split sur i=n-1, correct car 1≤n d'après P0,
     P2 }
   f.n + (s ↑ (↑i : i=n-1 ∧ f.i=A : S.i.(n-1)))
=  { S.(n-1).(n-1) = f.(n-1),
     max sur un domaine vide vaut −∞,
     alternative }
   if f.(n-1)≠A -> f.n + s
     |f.(n-1)=A -> f.n + (s ↑ f.(n-1))
   fi
```

D'où le programme :

```
  var A,B : int
; var f(0..N-1) : array of int {1≤N}
; var r,s : int

; n:=1 ; r:=-∞ ; s:=-∞
; do n≠N ->
    if f.(n-1)≠A -> s:= f.n + s
      |f.(n-1)=A -> s:= f.n + (s↑f.(n-1))
    fi
  ; if f.n≠B -> skip
      |f.n=B -> r:= r↑s
    fi
  ; n:=n+1
  od
```