Spécificité terminaison et correction de programme

Exercice 1

Exercice 1 On considère une fonction :

```
int find(int * t, int n, int x)
```

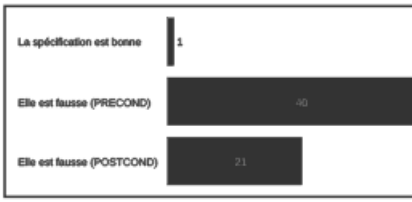
n peut être < 0.

qui cherche x dans le tableau t de taille n. La fonction `find` retourne la position i telle que $t[i] = x$ si x appartient à t et -1 sinon.

1. Pourquoi la spécification suivante ne correspond pas à celle de la fonction `find` (r est la valeur de retour de `find`) ?
   — PRECOND : $(n \in \mathbb{N})$
   — POSTCOND : $(r = -1) \vee (0 \leq r < n \wedge t[r] = x)$
2. Proposer une spécification correcte pour `find`.

1. Si find renvoie toujours -1, alors find satisfait la spécification.

2. - PRE $n \in \mathbb{N}$ (on peut aussi $n \geq 0$).
   - POST : $(0 \leq r < n \wedge t[r] = x)$
     $\vee$
     $(r = -1 \wedge \forall i \in [\![0, n-1]\!] : t[i] \neq x)$



Exercice 2

PRECOND :

POSTCOND :
malheureusement disparu !

Exercice 3

Exercice 3 Les deux algorithmes ci-dessous calculent la factorielle de n.

Version itérative :

```
1 algorithm fact (n) returns (f):
2   i := 1; f := 1;
3   while (i ≤ n) do
4     f := f*i;
5     i := i+1
6   end
```

Version récursive :

```
1 function fact_rec (n) returns (r):
2   if (n = 0) then
3     r := 1
4   else
5     r := n × fact_rec(n − 1)
6   end
```

Variant : $n - i + 1$

- n constant et i st croissant
  → $n - i + 1$ st déc.
- à valeurs dans $\mathbb{N}$
  * vrai en l.3
  * x l.4 $(n - i + 1 \in \mathbb{N} \wedge i \leq n)$
    ↓ nouvelle valeur de i
    l.6 $n - (i+1) + 1 \in \mathbb{N}$

Variant : n

- la suite des valeurs de n est st déc. pour appel récursif ✓
- à valeurs dans $\mathbb{N}$
  on appelle récursivement avec $n \neq 0$ et $n \in \mathbb{N}$
  → $n - 1 \in \mathbb{N}$ ✓

,,,,

,,,,,,,,

Exercice 4

Exercice 4 Les algorithmes suivants recherchent x dans le tableau t de taille n. Ils retournent *true* si x appartient à t et *false* sinon.

Version récursive :

```
1 function find_rec(x, t, n) returns (r):
2   if (n = 0) then
3     r := false
4   else if (t[n − 1] = x) then
5     r := true
6   else
7     r := find_rec(t, n − 1, x)
8   end
```

PRECOND: $n \in \mathbb{N}$, t est un tableau

POSTCOND: $r \Leftrightarrow \exists k,\ 0 \le k \le n-1 \wedge t[k] = x$

VARIANT : n est strictement ↘ par appel récursif. et $n \in \mathbb{N}$. Donc le programme termine.

Correction : Base : $n = 0$ t est vide et $x \notin t$ et $r = 0$. ✓

Induction : On suppose HI : find_rec $(t, n-1, x)$ est correcte.

On prend $n+1$:

→ Si $t[n-1] = x$ alors $r = true$ et $x \in t$ ✓

→ Sinon $t[n-1] \neq x$.

$r =$ find_rec $(t, n-1, x)$ Donc $x \in t$

$\Leftrightarrow x \in t[0, \dots, n-2]$ HI $\Leftrightarrow x \in t \wedge x \neq t[n-1]$

Or, $x \neq t[n-1]$. Donc :

$r \Leftrightarrow x \in t$. ✓

,,,,

Exercice 5 Les fonctions suivantes recherchent x dans le tableau t de taille n trié par ordre croissant. Elles retournent *true* si x appartient à t et *false* sinon. Comparer l'invariant à celui de l'exercice précédent.

```
1  algorithm binfind(x, t, n) returns (found):
2    l := 0; r := n-1;
3    p := (l + r) / 2;
4    while ((l ≤ r) ∧ (t[p] ≠ x)) do
5      if (t[p] < x) then
6        l := p + 1
7      else // (t[p] > x)
8        r := p - 1
9      end;
10     p := (l + r) / 2
11   end;
12   found := (l ≤ r)
```

PRECOND : t trié de taille n

POSTCOND : found $\Leftrightarrow \exists i \in [0;n[\,.\, t[i] = x$

VARIANT : $r - l + 1$

INVARIANT : $(\forall k \in [0;l[\,.\, t[k] \neq x)$
$\wedge (\forall k \in]r;n[\,.\, t[k] \neq x)$
$\wedge (l \leq r+1)$
$\wedge (l \leq p \leq r+1)$

,,,,,,,,

Exercice 5

Exercice 6

Terminaison variant l

— strictement décroissante par appel récursif / validé car tl est une sous liste de l

— à valeur dans un ensemble bien fondé / validé car $tl <_L hd :: tl \ \forall hd, tl$ est bien fondé sur le type *List* (théorème 7.3)

Correction
— PRECOND : $l \in List$
— POSTCOND : $is\_sorted(l) \iff (\forall i \in [0; |l|[. l_i \leq l_{i+1})$

Cas de base :
si $l = [] \rightarrow is\_sorted(l) = true$ validé !
si $l = hd :: [] \rightarrow is\_sorted(l) = true$ validé !
Induction : On suppose l'appel récursif correct, (HI, la ligne 5)
$hd :: \underbrace{hd_2 :: tl_2}_{\text{tl}}$ il faut que tl trié $\leftarrow bonparHI \implies is\_sorted(l)$ correct
.