Division par deux, correcte

http://www.enseignement.polytechnique.fr/profs/informatique/Luc.Maranget/IF/10/

Soit `S` une instruction, l'écriture $\{P\}$ `S` $\{Q\}$, où P est la pré-condition et Q la post-condition, peut se comprendre comme, si P est vraie avant l'exécution de `S` alors Q est vraie après l'exécution de `S`. Dans les programmes on écrit les propositions entre les instructions.

Sans être complètement formel, il semble clair :

- Que $\{P\}$ `int x = 0` $\{x = 0 \land P\}$ est raisonnable.
- Que $\{P\}$ `S; S'` $\{Q\}$ se déduit de $\{P\}$ `S` $\{R\}$ et $\{R\}$ `S'` $\{Q\}$.
- Que de même, on peut déduire $\{P\}$ `if (E) S else S'` $\{Q\}$ de $\{P \land E\}$ `S` $\{Q\}$ et de $\{P \land \neg E\}$ `S'` $\{Q\}$.
- Enfin, étant donnée une boucle `while (E) S`, on appelle invariant de boucle une proposition I, telle que $\{I \land E\}$ `S` $\{I\}$. Dès lors, on aura $\{P\}$ `while (E) S` $\{Q\}$ quand il existe in invariant I et que l'on a en outre $P \Rightarrow I$ et $(I \land \neg E) \Rightarrow Q$.

1 La recherche dichotomique

Le principe de la recherche dichotomique dans un tableau trié est bien connu.

1. Expliquer comment ce principe permet de deviner un nombre choisi au hasard entre 1 et 10 en au plus 4 étapes. Le joueur qui cherche propose un nombre et l'autre répond trop grand, trop petit ou exact.
2. Voici une méthode de recherche d'un nombre k dans un tableau `t[`$0 \ldots n$`[`.

```
static boolean search(int [] t, int k) {
  int low = 0 ;
  int high = t.length ;
  while (low < high) {
    int m = (low+high) / 2 ;
    if (k < t[m]) {
      high = m ;
    } else if (k > t[m]) {
      low = m+1 ;
    } else {
      return true ;
    }
  }
  return false ;
}
```

   Équiper le code ci-dessus avec des assertions qui en prouvent la correction. On utilisera la propriété $\phi(i, j)$ qui exprime que si k se trouve dans le tableau complet `t[`$0 \ldots n$`[`, alors k se trouve dans le sous-tableau `t[`$i \ldots j$`[`.

$$\phi(i, j) = (k \in \mathtt{t[}0 \ldots n\mathtt{[} \Rightarrow k \in \mathtt{t[}i \ldots j\mathtt{[})$$

   On commencera par trouver l'invariant en n'oubliant pas une condition de validité des indices.
3. Prouver la terminaison du programme.
4. Que va t-il se passer si le tableau `t` n'est pas trié ? Et dans le cas particulier d'un tableau trié à l'envers ?
5. Modifier le programme pour utiliser des tableaux fermés `t[`$0 \ldots n-1$`]`.

2 Diviser par deux, en plus risqué

Voici une version finement optimisée de la recherche dichotomique, munie d'un début d'invariant de boucle et d'une assertion importante. Notez que l'on se donne deux éléments hypothétiques t[−1] et t[n], avec t[−1] $< k \leq$ t[n]).

```
  static boolean searchBis(int [] t, int k) {
    int low = -1 ;
    int high = t.length ;
// t[low] < k ≤ t[high]
    while (high-low > 1) {
      int m = (low+high)/2 ;
      if (t[m] < k) {
        low = m ;
      } else
        high = m ;
    }
// high = low + 1 ∧ t[low] < k ≤ t[high]
    return high < t.length && t[high] == k ;
  }
```

1. Où est l'optimisation ?
2. Comment l'assertion finale garantit-elle la correction de la réponse.
3. Revisiter l'argument de terminaison.
4. Prouver franchement la correction en donnant les assertions.

3 La fonction 91

La technique des assertions peut également s'employer pour les fonctions récursives. Les preuves de corrections fonctionnent en deux temps.

- On énonce la propriété de correction et on démontre que supposer la propriété au sujet des appels (récursifs) entraîne sa validité pour l'appel courant.
- On montre la terminaison.

Dans cet exercice on considère la fonction suivante.

```
  static int f(int x) {
    int r ;
    if (x > 100)
      r = x-10 ;
    else
      r = f(f(x+11)) ;
    return r ;
  }
```

Et on se propose de démontrer :

$$f(x) = \texttt{if } x > 100 \texttt{ then } x - 10 \texttt{ else } 91$$

Que l'on note $\mathcal{P}(x, f(x))$.

1. Montrer la correction partielle, c'est-à-dire fermer le chemin entre les deux assertions suivantes.

```
    static int f(int x) {
      int r ;
  // ∀x.P(x, f(x))
```

```
    if (x > 100) {
      r = x-10 ;
    } else {
      r = f(f(x+11)) ;
    }
// 𝒫(x,r)
    return r ;
  }
```

2. Montrer la terminaison. Pour y arriver on utlisera l'ordre bizarre suivant, $-\infty \succ \ldots \succ 98 \succ 99 \succ 100$ et tous les entiers strictement supérieurs à 100 sont incomparables entre eux mais $\prec$ aux entiers inférieurs ou égaux à 100. Sans en rajouter dans le formalisme, on peut considérer que la correction partielle est l'étape inductive d'une preuve. Il suffit maintenant de montrer que les appels (récursifs) s'exercent sur des arguments décroisssants selon $\prec$.

Solutions des exercices

Exercice 1

1. Quand on sait que le nombre demandé est entre x et y, on propose la moyenne $m = (x+y)/2$. En cas d'échec, on continue la recherche dans les intervalles $[x \ldots m-1]$ ou $[m+1 \ldots y]$ selon que m est plus grand ou plus petit que le nombre mystère. Dans le cas de l'intervalle de départ $[1 \ldots 10]$ on propose 5, ce qui nous donne en cas d'échecs répétées un intervalle de taille au pire 5, puis 2, et c'est fini (intervalle de taille 1 et donc réponse exacte assurée, ou détection, au mieux, d'une erreur dans les réponses précédentes).
2. Soit la propriété $\phi(i,j)$ qui exprime que si k se trouve dans le tableau complet $\mathtt{t[}0\mathtt{...}n\mathtt{[}$, alors k se trouve dans le sous-tableau $\mathtt{t[}i\mathtt{...}j\mathtt{[}$.

$$\phi(i,j) = (k \in \mathtt{t[}0\mathtt{...}n\mathtt{[} \Rightarrow k \in \mathtt{t[}i\mathtt{...}j\mathtt{[})$$

```
  static boolean search(int [] t, int k) {
    int low = 0 ;
    int high = t.length ;
// 0 ≤ low ≤ high ≤ n ∧ φ(low, high)
    while (high-low > 0) {
// 0 ≤ low < high ≤ n ∧ φ(low, high)
      int m = (low+high) / 2 ;
// 0 ≤ low ≤ m < high ≤ n ∧ φ(low, high)
      if (k < t[m]) {
// φ(low, m) ∧ 0 ≤ low ≤ m < high ≤ n
        high = m ;
// φ(low, high) ∧ 0 ≤ low ≤ high < n
      } else if (k > t[m]) {
// φ(m + 1, high) ∧ 0 ≤ low ≤ m < high ≤ n ∧ φ(low, high)
        low = m+1 ;
// φ(low, high) ∧ 0 < low ≤ high ≤ n
      } else {
// t[m] = k
        return true ;
// Rien ~!
      }
// φ(low, high) ∧ 0 ≤ low ≤ high ≤ n
    }
// 0 ≤ low = high ≤ n ∧ φ(low, high)
    return false ;
  }
```

La correction est prouvée car la dernière assertion entraîne que k n'est pas dans le tableau. Les passages importants sont de $\phi(\mathit{low}, \mathit{high})$ à $\phi(\mathit{low}, m)$ et $\phi(m+1, \mathit{high})$, ils utilisent l'hypothèse d'un tableau trié.

3. Il y a un argument qui tue : entre le début et la fin d'une itération de la boucle, un élément (celui d'indice m) est exclu de l'intervalle de recherche. Cet argument révèle la robustesse du programme, m peut aussi bien être n'importe quel indice valide de l'intervalle $[\mathit{low} \ldots \mathit{high}[$.
Plus conformément au cours, on peut considérer l'ordinal $\mathit{high} - \mathit{low}$ (c'est bien un entier naturel) qui décroit strictement du début à la fin d'une itération. En effet sous l'hypothèse $\mathit{low} < \mathit{high}$ et en posant $m = (\mathit{low} + \mathit{high})/2$ alors il vient $\mathit{low} \le m < \mathit{high}$. Soit encore $m - \mathit{low} < \mathit{high} - \mathit{low}$ et $\mathit{high} - m \le \mathit{high} - \mathit{low}$ (et donc $\mathit{high} - (m+1) < \mathit{high} - \mathit{low}$). On

peut être plus précis, si $high - low$ est impair et vaut $2p+1$ alors le nouvel intervalle est de taille p, sinon $high - low = 2p$ ($p \geq 1$) et le nouvel intervalle est de taille p ou $p-1$.

4. Un premier point important est de constater que la terminaison ne dépend pas des éléments du tableau. En revanche la correction en dépend. Dans tous les cas, si `search(t, k)` répond vrai, alors le résultat est exact, sinon, le résultat peut être exact. Dans le cas où le tableau est trié à l'envers, on aura `search(t, k) == true`, si et seulement si `k` est l'élément d'indice médian de `t`.
5. Le code est peut être un peu plus régulier :

```java
static boolean search(int [] t, int k) {
  int low = 0 ;
  int high = t.length-1 ;
  while (low <= high) {
    int m = (low+high) / 2 ;
    if (k < t[m]) {
      high = m-1 ;
    } else if (k > t[m]) {
      low = m+1 ;
    } else {
      return true ;
    }
  }
  return false ;
}
```

Exercice 2

1. Il y a une seule comparaison au lieu de deux dans le corps de la boucle. On peut espérer un gain non négligeable. La « preuve » d'un tel bout de code devient un impératif.
2. C'est évident en cas de réponse positive, (à cause du test d'égalité). C'est un peu moins direct en cas de réponse négative. Supposons $-1 \leq low < high \leq n$ (voir la suite, mais il est assez évident que *low* croît et *high* décroit, tous deux au sens large) et distinguons trois sous-cas,
   - Si $low = -1$ alors *high* est zéro et k est strictement plus petit que tous les éléments du tableau qui est trié.
   - Si $high = n$, alors *low* est $n-1$ et k est strictement plus grand que tous les éléments du tableau.
   - Sinon, on a trouvé $x = low$ indice valide, tel que $x+1$ est aussi un indice valide, avec $\mathtt{t}[x] < k < \mathtt{t}[x+1]$, et donc k ne peut pas se trouver dans le tableau `t`.

   Il est notable que c'est la seule intervention de l'hypothèse de tri.
3. Sous l'hypothèse $high - low > 1$, on a cette fois $low < m < high$. La quantite $high - low$ décroit strictement, d'un tour de boucle au suivant. Plus précisément, la quantité $high - low$ passe de $2p+1$ (avec $1 \leq p$) à p ou $p+1$, ou bien de $2p$ (avec $1 \leq p$) à p. Il y a donc bien décroissance stricte.
4. Enrichissons l'invariant de quelques inégalités et ajoutons les assertions.

```java
  static boolean searchBis(int [] t, int k) {
    int low = -1 ;
    int high = t.length ;
// −1 ≤ low < high ≤ n ∧ t[low] < k ≤ t[high]
    while (high-low > 1) {
// −1 ≤ low < low + 1 < high ≤ n ∧ t[low] < k ≤ t[high]
      int m = (low+high)/2 ;
```

```
// $-1 \leq low < m < high \leq n \land$ t[$low$] $< k \leq$ t[$high$]
      if (t[m] < k) {
// $-1 \leq low < m < high \leq n \land$ t[$m$] $< k \leq$ t[$high$]
        low = m ;
// $-1 \leq low < high \leq n \land$ t[$low$] $< k \leq$ t[$high$]
      } else
// $-1 \leq low < m < high \leq n \land$ t[$low$] $< k \leq$ t[$m$]
        high = m ;
// $-1 \leq low < high \leq n \land$ t[$low$] $< k \leq$ t[$high$]
    }
// $-1 \leq low < high \leq n \land high = low + 1 \land$ t[$low$]$k \leq$ t[$high$]
    return high < t.length && t[high] == k ;
  }
```

Une fois trouvé l'invariant la pose des insertions est assez mécanique.

Exercice 3

1. On commence par mettre les assertions c'est assez facile.

```
  static int f(int x) {
    int r ;
// $\forall x.\mathcal{P}(x, f(x))$
    if (x > 100) {
// $x > 100$
      r = x-10 ;
// $x > 100 \land r = x - 10$
    } else {
// $x \leq 100 \land \forall x.\mathcal{P}(x, f(x))$
      r = f(f(x+11)) ;
// $x \leq 100 \land r = 91$
    }
// $\mathcal{P}(x, r)$
    return r ;
  }
```

Le passage difficile est de $x \leq 100 \land \forall x.\mathcal{P}(x, f(x))$ à $x \leq 100 \land r = 91$. Par instanciantion (remplacer x par $x + 11$) on a

$$f(x + 11) = \texttt{if } x + 11 > 100 \texttt{ then } x + 11 - 10 \texttt{ else } 91$$

D'où il vient, par quelques calculs et par $f(\texttt{if } e \texttt{ then } e_t \texttt{ else } e_f)$ égal à $\texttt{if } e \texttt{ then } f(e_t) \texttt{ else } f(e_f)$

$$f(f(x + 11)) = \texttt{if } x > 89 \texttt{ then } f(x + 1) \texttt{ else } f(91)$$

On instancie deux fois $\forall x.\mathcal{P}(x, f(x))$ un fois par $x + 1$ et l'autre par 91.

$$f(f(x + 11)) = \texttt{if } x > 89 \texttt{ then } (\texttt{if } x + 1 > 100 \texttt{ then } x + 1 - 10 \texttt{ else } 91) \texttt{ else } 91$$

Soit en tenant compte enfin de l'hypothèse $x \leq 100$.

$$f(f(x + 11)) = \texttt{if } x > 89 \texttt{ then } (\texttt{if } x = 100 \texttt{ then } 100 + 1 - 10 \texttt{ else } 91) \texttt{ else } 91$$

2. Il faut donc montrer $x + 11 \prec x$ et $f(x + 11) \prec x$ sous l'hypothèse $x \leq 100$.
   La propriété $x + 11 \prec x$ est facile par définition de l'ordre, et ceci nous donne le droit d'utiliser la proposition $\mathcal{P}(x + 11, f(x + 11))$, dans le sens que nous savons maintenant que f termine pour l'argument $x + 11$.

- Donc si $x+11 \leq 100$ (c'est à dire $x \leq 89$) on a $f(x+11)=91$. Et on a bien $f(x+11) \prec x$ (car $x \leq 89 < 91$),
- Sinon $x+11 > 100$ et $f(x+11) = x+11-10 = x+1$ et l'on peut encore conclure puisque $x+1 > x$.