

Objectifs

- À la fin de cette séance, vous devriez être capable de :
- proposer et représenter un invariant de boucle ;
 - spécifier un algorithme récursif ;
 - prouver la correction d'un algorithme récursif simple ;
 - prouver la terminaison d'un algorithme récursif.

Exercice 1 : Dessins d'invariants

Rappel : un invariant est une propriété qui est maintenue par chaque itération : si elle est vraie au début d'une itération quelconque, alors elle est encore vraie à la fin de cette itération.

On s'intéresse à l'algorithme suivant :

```
Maximum(T[0..n - 1]) :
  /* Postcondition : max est une valeur ≥ à toutes celles dans T */
  max := T[0]
  pour i := 1 à n - 1 :
    si T[i] > max :
      max := T[i]
```

1. Proposez un ou plusieurs invariants et représentez-les graphiquement.
2. On rappelle qu'un invariant n'est utile qu'à condition :
 - qu'il soit vrai au moment où on entre dans la boucle
 - qu'il permette de déduire la postcondition en sortie de boucle
 Parmi les invariants que vous avez proposés, lesquels ne vérifient pas tous ces critères ?
3. Si besoin, proposez un nouvel invariant qui vérifie tous les critères cités ci-dessus.
4. Une fois que vous avez obtenu un invariant utile, démontrez qu'il s'agit bien d'un invariant.
5. Mêmes questions pour l'algorithme :

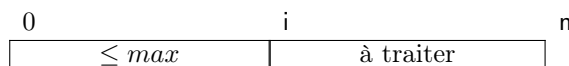
```
Minimum(T[0..n - 1]) :
  /* Postcondition : T[i] est le minimum de T */
  i := 0
  j := n - 1
  tant que i < j :
    si T[i] < T[j] :
      j := j - 1
    sinon :
      i := i + 1
```

6. *Bonus, à ne faire qu'après l'exercice 2* : mêmes questions pour ce troisième algorithme.

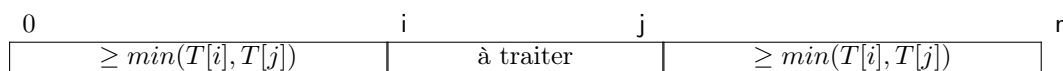
```
Pangramme(T[0..n - 1], k) :
  /* Postcondition : renvoie vrai si le tableau T contient au moins un
   *   exemplaire de chaque nombre entre 0 et k */
  Soit V un tableau de booléens indexé de 0 à k.
  pour j := 0 à k :
    V[j] := faux
  pour i := 0 à n - 1 :
    V[T[i]] := vrai
  pour j := 0 à k :
    si (non V[j]) :
      renvoyer faux
  renvoyer vrai
```

Correction de l'exercice 1

1. Toutes les valeurs dans $T[0..i - 1]$ sont $\leq max$.
Attention à être précis sur l'indice, y compris sur le schéma ! En particulier, un indice se positionne toujours sur une cellule, pas à la frontière entre deux cellules.



2. Cet invariant est suffisant pour la postcondition énoncée : il est vrai initialement (car il ne concerne aucune valeur quand $i = 0$), et en sortie de boucle $i = n$ donc $max \geq T[0..n - 1]$.
Remarquons cependant que pour un calcul de maximum, il faudrait également que max corresponde à une des valeurs de T , ce qu'on peut montrer à part dans un second temps.
Quelques exemples d'invariants insatisfaisants :
 - $max \geq T[0..i]$: ce n'est pas un invariant (on ne sait rien sur max et $T[i + 1]$ donc on ne peut pas le prouver pour l'itération suivante).
 - $max \geq T[0..n - 1]$: c'est un invariant (la propriété ne dépend pas de i donc elle garde la même valeur de vérité à toutes les itérations) mais il n'est pas vrai en entrant dans la boucle.
 - $i \leq n$ ou bien $max \geq T[0]$: c'est vrai en entrant dans la boucle et c'est un invariant, mais il ne permet pas de montrer la postcondition.
3. cf question 2.
4. Ici l'invariant pourrait être : toutes les valeurs dans $T[0..i - 1]$ sont supérieures à $\min(T[i], T[j])$, et de même pour les valeurs dans $T[j + 1..n - 1]$.



À nouveau cela ne concerne aucune valeur initialement.

Il est maintenu par une itération car on intègre la plus grande valeur parmi $T[i]$ et $T[j]$ dans $T[0..i - 1] \cup T[j + 1..n - 1]$.

Appelons m le $\min(T[i], T[j])$ en début d'itération. Le nouveau $T[i']$ ou $T[j']$ est soit supérieur à m et donc l'invariant est facilement maintenu ; ou bien il est $< m$ et a fortiori les valeurs dans $T[0..i - 1] \cup T[j + 1..n - 1]$ sont $> m > T[i']$.

Enfin, en sortie de boucle, $i = j$ donc $T[i]$ est inférieur à toutes les autres valeurs de T .

5. La boucle importante à étudier est la deuxième.
L'invariant est ici : pour tout indice j compris entre 0 et k , la cellule $V[j]$ contient *vrai* si et seulement si j est présent dans $T[0..i - 1]$.
En sortie de boucle V nous renseigne donc précisément sur les valeurs contenues dans T .

Exercice 2 : Preuve d'algorithme récursif

On a écrit l'algorithme suivant pour déterminer si un arbre binaire est *parfait*, c'est-à-dire que toutes ses feuilles sont à la même profondeur, et tous ses autres nœuds ont deux fils non vides.

```

Parfait(A) :
  si A est une feuille :
    └ renvoyer 0
  sinon si FilsGauche(A) est vide ou FilsDroit(A) est vide :
    └ renvoyer -1
  sinon :
    hg := Parfait(FilsGauche(A))
    hd := Parfait(FilsDroit(A))
    si hg ≥ 0 et hd ≥ 0 et hg = hd :
      └ renvoyer hg + 1
    sinon :
      └ renvoyer -1
  
```

On cherche à démontrer que cet algorithme récursif est correct.

1. Dessinez à quoi ressemble un arbre parfait.
2. Écrivez la spécification (précondition, postcondition) de la fonction récursive Parfait.
3. Vérifiez que les appels récursifs vérifient votre precondition.
4. Démontrez que les cas de base vérifient votre postcondition.
5. Démontrez que les cas récursifs vérifient votre postcondition.
(Rappel : vous pouvez pour cela supposer que les appels récursifs sont corrects.)
6. Démontrez la terminaison de cette fonction.

Correction de l'exercice 2

1. Il contient toujours $2^{h+1} - 1$ nœuds, dont 2^h feuilles.
Remarquons que dans un arbre parfait, tous les sous-arbres sont aussi parfaits (mais cela ne constitue pas une condition suffisante).
2. — Précondition : A est non vide (criticable, on pourrait considérer qu'un arbre vide est parfait).
— Postcondition : renvoie la hauteur de A si A est parfait, ou -1 dans les autres cas.
3. Oui, on s'assure que $FG(A)$ et $FD(A)$ sont non vides avant de faire les appels récursifs.
4. Il y a deux cas à considérer :
— si A est une feuille, alors il est bien parfait et sa hauteur est 0.
— dans le second cas du If, on sait que un des fils de A est vide mais pas l'autre (sinon c'est une feuille). Il n'est donc pas parfait et on renvoie -1.
5. — Si un des appels récursifs renvoie -1, alors A n'est pas parfait, et on renvoie bien -1 à notre tour.
— Si les deux appels récursifs renvoient une hauteur différente, alors A n'est pas parfait non plus (il a des feuilles à la hauteur hg et d'autres à la hauteur hd), et on renvoie bien -1 à nouveau.
— Si les deux appels récursifs renvoient la même hauteur, alors A est parfait (toutes ses feuilles sont à la hauteur $hg = hd$, sa racine est complète, et par hypothèse tous les nœuds de ses sous-arbres sont complets).
On renvoie alors bien la hauteur de A .
6. La hauteur des arbres diminue dans les appels récursifs.