Validation d'algorithmes
L3 MIAGE, UGA
2020/2021

Raphaël Charrondière
raphael.charrondiere@inria.fr
Semaine 3 - échéance S4

Feuille de TD 2: Preuves suite

Où l'on continue à manipuler les variants, les invariants, les ordres bien fondés.

Exercice 1 Tri bulle

L'un des algorithmes de tri facile à mettre en place est le tri bulle (algorithme 1). En parcourant la liste, on intervertit les nombres consécutifs qui ne sont pas dans le bon ordre, puis on recommence jusqu'à ce que la liste ne soit plus modifiée.

La difficulté ici est dû au fait des boucles imbriquées. Il va d'abord falloir étudier la boucle interne (ligne 5 à 10) en premier et en déduire une propriété. On admet qu'elle termine (Il faudrait formaliser que $n-i$ diminue strictement et reste positif à chaque entrée de boucle ligne 5. Pour une boucle for simple comme celle-là, ça n'a pas grand intérêt.) Il faut trouver une propriété exploitable.

1 Quel ordre bien fondé pourrait être utilisé sur les listes pour montrer qu'une liste est mieux triée qu'une autre ?

2 Formuler la propriété exploitable du bloc interne (lignes 5 à 10)

3 Prouver que l'algorithme 1 termine et renvoie le bon résultat. Cette étape doit être intuitive si la question précédente à bien été répondue, sinon, essayez de modifier votre propriété à la question précédente.

```
   Entrées : l liste d'entiers
   Output : la liste l triée par ordre croissant.
 1 stop ← faux;
 2 n ← length(l);
 3 tant que ¬stop faire
 4 |   stop ← vrai;
 5 |   pour i de 1 à n − 1 faire
 6 |   |   si l[i − 1] > l[i] alors
 7 |   |   |   intervertir(l[i − 1], l[i]);
 8 |   |   |   stop ← faux;
 9 |   |   fin
10 |   fin
11 fin
12 retourner r
```

Algorithme 1 : Tri bulle

```
   Entrées : i ∈ ℕ
   Output : i
1 si i == 0 alors
2 |   retourner 0;
3 sinon
4 |   retourner 1+StupidAdd(i-1);
5 fin
```

Algorithme 2 : StupidAdd

```
   Entrées : t un arbre binaire de recherche, e
             un élément
   Output : e appartient-il à t ?
 1 r ←root(t);
 2 tant que r est défini faire
 3 |   si r<e alors
 4 |   |   r← filsDroite(t);
 5 |   sinon si r>e alors
 6 |   |   r← filsGauche(t);
 7 |   sinon
 8 |   |   retourner Vrai;
 9 fin
10 retourner Faux;
```

Algorithme 3 : Recherche dans un ABR

Exercice 2 Et en mode récursion ?

Les exemples avec invariant ont pour l'instant fait intervenir une boucle while. Malheureusement dans certains cas, il est plus facile de coder une fonction récurrente, i.e. qui s'appelle elle-même, on va donc devoir s'adapter. Supposons qu'une fonction f ne contienne que des appels à elle même et des instructions qui terminent, et des conditions.

1 Considérer la fonction StupidAdd (Algorithme 2). Pourquoi termine-t-elle ?

Raphaël Charrondière
raphael.charrondiere@inria.fr

2➢ Proposer une méthode générique pour montrer que f termine, potentiellement avec une condition sur l'état mémoire en entrée. *Remarque : Inspirez vous fortement du cours. Aide : qu'est-ce que voudrait-dire que f ne termine pas*

3➢ Considérer la fonction StupidAdd (Algorithme 2). Pourquoi renvoie-t-elle ce qu'il faut (Pourquoi StupidAdd(i)=i) ?

4➢ En supposant que f termine, proposer une méthode générique pour montrer que f «fait bien ce qu'il faut» c.-à-d. est correcte. *Remarque : Commencez avec le cas où f ne s'appelle qu'une fois maximum et pensez au cas où f ne s'appelle pas.*

Exercice 3 — Arbres binaires de recherche

Reprenons les arbres binaires du cours. L'intérêt d'un arbre binaire est de pouvoir recherche rapidement un élément. On propose une fonction dans l'algorithme 3

1➢ Créez un ABR équilibré (de profondeur minimale) contenant les éléments 1, 4, 8, 15, 16, 46, 72, 73, 77, 144, 154 et 160. Ensuite détaillez l'éxécution de l'algorithme 3 sur l'entrée $e = 5$ et $e = 78$.

2➢ Donnez un invariant et un variant qui prouveraient la terminaison et la correction

3➢ Réécrivez la fonction en récursif. *Utilisez une structure similaire à l'algorithme 2 si vous ne savez pas comment faire.*

4➢ Reprenez les méthodes que vous avez proposé dans l'exercice 2 pour prouver la terminaison et correction de la nouvelle version de l'algorithme.

L'intérêt de la structure ABR est la facilité d'insérer des éléments dynamiquement. Son inconvénient est le déséquilibrage facile de la structure. On peut parfois rééquilibre l'ABR. Si vous voulez, vous pouvez écrire la fonction de suppression d'un élément puis prouver sa terminaison et correction.