Projet : Calcul du segment majoré en temps linéaire

Yann Régis-Gianas (yrg@pps.jussieu.fr)
GL6 - Licence 3 - Université Paris 7

1 Théorie des piles (5 points)

Dans cette partie, on définit une théorie pour les **piles d'entiers** en les traitant comme des séquences finies d'entiers.

1.1 Constructeurs / Destructeurs de pile et leurs axiomes

Voici une définition en langage naturel des valeurs et des opérations définissant les piles :

« Une pile est soit vide (on la note "empty"), soit composée d'un entier x situé à son sommet et sous lequel on trouve une pile S (on la note alors "cons (x, S)"). Par exemple, le terme S valant "cons (0, cons (2, cons (1, empty)))" est une pile au sommet de laquelle on trouve l'entier 0*. Au dessous de cet entier se trouve la pile S' valant "cons (2, cons (1, empty))". L'opération "top" extrait le sommet d'une pile lorsqu'il existe. L'opération "push (0, S')" vaut S et l'opération "pop (S)" vaut S'. »*

❶ Définissez le type stack de piles d'entiers ainsi que les symboles empty et cons.

❷ Quels axiomes sont nécessaires pour définir correctement l'égalité entre deux piles ?

❸ Déclarez les opérations logiques top, push et pop puis spécifiez les propriétés de ces opérations à l'aide d'axiomes.

❹ Prouvez les propriétés liant les piles S et S' définies plus haut.

2 Implémentation d'une structure de pile bornée (5 points)

On se propose de fournir une implémentation de la structure de pile en utilisant un entier n et un tableau **fonctionnel** d'entiers t. L'entier n représente la hauteur de la pile et le tableau t les éléments qui la composent. Ainsi, le sommet de la pile est contenu dans la case $n-1$ du tableau.

2.1 Primitives supplémentaires sur les tableaux

Pour simplifier l'implémentation, notre structure de donnée sera fonctionnelle : chaque opération appliquée sur une pile en produit une nouvelle.

❶ Donnez le type et la spécification d'une fonction farray_new qui attend deux entiers N et x et qui produit un tableau fonctionnel de taille N dont chaque cellule vaut x.

2.2 Pile représentée par un tableau et un entier

❶ Définissez le type stacki des implémentations de pile et trois opérations :
– "mk_stacki" qui produit une implémentation de pile à partir d'un entier et d'un tableau fonctionnel.
– "stacki_length" qui extrait la taille d'une implémentation de pile.
– "stacki_values" qui extrait les valeurs contenues dans la pile.

❷ Quelles contraintes sont nécessaires à la bonne formation d'une telle implémentation de pile ? Définissez un prédicat dénotant cette propriété.

❸ Comment définir l'égalité entre deux implémentations de pile ?

2.3 Opérations

❶ À l'aide d'axiomes, définissez une fonction d'interprétation "stackof" de type stacki → stack qui associe une implémentation de pile à la pile « logique » qu'elle représente.

❷ Définissez, spécifiez et vérifiez les opérations suivantes sur les implémentations de pile :
- "stacki_empty" qui produit une pile pouvant contenir au plus N éléments.
- "stacki_is_empty" qui calcule un booléen valant true si et seulement si une implémentation représente la pile vide.
- "stacki_push" qui implémente l'opération push sur la pile représentée par l'implémentation dans la limite de la capacité de l'implémentation.
- "stacki_top" qui implémente l'opération top sur la pile représentée par l'implémentation.
- "stacki_pop" qui implémente l'opération pop sur la pile représentée par l'implémentation.

3 Calcul naïf du segment majoré (5 points)

3.1 Problème

Soit un tableau d'entiers t, représentant une série de valeurs indicées par le temps, et un indice i compris dans les bornes de ce tableau. Quel est le nombre N_i d'indices consécutifs jusqu'à i tels que les valeurs de ces indices sont plus petites ou égales à celle de l'indice i ? Le segment $[i - N_i; i]$ est appelé **segment majoré par** i.

Voici, en guise d'illustration, l'évolution du prix du *Pan Galactic Gargle Blaster* pendant les 6 premiers siècles de notre ère :

0	1	2	3	4	5
3	5	2	6	7	3

Pour chaque indice i, on calcule un nouveau tableau contenant les valeurs N_i :

0	1	2	3	4	5
1	2	1	4	5	1

On peut alors dire, par exemple, que « le prix durant le siècle numéro 3 n'avait jamais été aussi élevé depuis 4 siècles. » L'objectif des deux sections suivantes est de calculer efficacement ce dernier tableau.

3.2 Description informelle de l'algorithme

❶ Imaginez un algorithme (le plus simple possible) qui calcule, pour un indice i et un tableau t donné l'entier N_i défini informellement dans la section précédente. Écrivez-le en pseudo-code.

❷ Exécutez votre algorithme sur l'entrée suivante :

0	1	2	3	4	5
5	1	2	3	7	1

3.3 Spécification

❶ Complétez la définition du prédicat span_correct suivante :

```
predicate span_correct (src: int farray, span : int farray, i : int) =
    ...
```

Ce prédicat signifie qu'à la case i du tableau span se trouve l'entier N_i défini précédemment.

❷ À l'aide du prédicat précédent, donnez une spécification à l'algorithme de calcul du segment majoré.

3.4 Algorithme

❶ Développez l'implémentation de votre algorithme à l'aide de Why, en lui donnant la spécification définie dans la question précédente. La signature de l'algorithme devra être :

```
let naive_spanning (src: int array) (span : int array) =
  ...
```

❷ Quelles obligations de preuve sont d'ores et déjà validées par Why ?

3.5 Invariant de boucles

❶ Décrivez informellement le fonctionnement des boucles du programme.

❷ Décrivez informellement l'invariant de chacune de ces boucles.

❸ Terminez la vérification du programme Why.

3.6 Complexité

❶ Quelle est la complexité de votre algorithme en termes de nombre de comparaisons ?

❷ Quelles sont les raisons de son inefficacité ?

❸ Quelles propositions pourriez-vous faire pour l'améliorer ?

4 Calcul du segment majoré en temps linéaire (5 points)

Dans cette dernière partie, on s'intéresse à un algorithme linéaire de calcul du segment majoré qui s'appuie sur une structure de pile.

4.1 Algorithme linéaire

L'idée de l'algorithme linéaire est de maintenir une pile des indices « intéressants » (à vous de trouver pourquoi) et de minimiser ainsi le nombre de comparaisons nécessaires au calcul. On donne l'algorithme sous la forme d'un programme Why :

```
let spanning (src: int array) (span : int array) =
  let i = ref 0 in
  let s = ref (stacki_empty (array_length (src))) in
    while ( !i < array_length_ (src)) do
      while (not stacki_is_empty !s) && src[stacki_top !s] ≤ src[ !i] do
        s := stacki_pop !s
      done;
      if stacki_is_empty !s then
        span[ !i] := !i + 1
      else
        span[ !i] := !i - stacki_top !s;
      s := stacki_push !i !s;
      i := !i + 1
    done
```

❶ Exécutez cet algorithme sur les deux entrées dans la section 3.2. Faites apparaître l'évolution de la pile au fur et à mesure de l'exécution.

❷ Expliquez informellement votre compréhension du fonctionnement de cet algorithme.

4.2 Spécification faible

❶ Sans spécifier la précondition et la postcondition pour le moment, exécutez Why sur votre programme. À quoi correspondent les obligations de preuve engendrées par Why ?

❷ Déterminez les invariants de boucle nécessaires à la validation des obligations de preuve produites par Why.

4.3 Spécification complète

❶ Donnez à cet algorithme la spécification explicitée dans la section 3.3.

❷ Sans renforcer les invariants de boucle, quelles nouvelles obligations de preuve sont prouvées ? Lesquelles ne le sont pas ?

❸ Quelles sont selon vous les propriétés importantes qui devront apparaître dans les invariants de boucle de façon à justifier la correction de cet algorithme ?

4.4 Invariant de bonne formation de la pile

❶ Dans quelle situation la pile peut-elle être vide ?

❷ Lorsque la pile n'est pas vide, quelle relation peut-on établir vérifier à tout moment entre l'indice situé au sommet de la pile et l'indice i ?

❸ En déduire un prédicat wf liant la pile et le tableau d'entrée et correspondant aux propriétés de la pile maintenues constamment par l'algorithme et sur lesquelles il s'appuie.

```
logic wf : int farray, int stack → prop
```

4.5 Invariants des boucles

❶ Utilisez le prédicat précédent pour préciser la boucle externe du programme.

❷ Utilisez le prédicat précédent pour préciser la boucle interne du programme.

❸ Vérifiez la correction de l'algorithme. N'hésitez pas à décomposer chaques étapes du raisonnement et à faire apparaître des lemmes et assertions intermédiaires.

5 Travail à rendre

Votre travail sera composé de 5 fichiers :
- `stack.mlw` contenant la théorie des piles.
- `stacki.mlw` contenant l'implémentation de la structure de pile.
- `spanning-naive.mlw` contenant l'implémentation et la vérification de l'algorithme naïf de calcul du segment majoré.
- `spanning-linear.mlw` contenant l'implémentation et la vérification de l'algorithme linéaire de calcul du segment majoré.
- `rapport.pdf` un document contenant les indications nécessaires à la compréhension de votre démarche pour chacune des questions.

L'ensemble de ces fichiers devra être rendu sous la forme d'une archive nommée `prenom-nom.tar.gz`. Cette archive devrait être envoyée par e-mail avant le 27 avril 2009. Votre projet sera évalué lors d'une soutenance individuelle.