Matthieu Boyer
Planche A

Question de Cours

Donnez une définition du type `int list` en OCaml. Quelles sont les complexités des opérations suivantes :

1. Trouver un élément dans une liste ;
2. Ajouter un élément au milieu de la liste ;
3. Supprimer le premier élément de la liste.

DLL

On écrira le code de cet exercice en OCaml. On donnera toujours la complexité des différentes fonctions écrites.

On s'intéresse à une structure de données σ ordonnée, de sorte qu'on puisse ajouter et supprimer un élément au début ou à la fin de σ.

1. Donnez une définition possible d'une telle structure.
2. Donnez une fonction créant une telle structure à partir des éléments d'une liste en OCaml.
3. Donnez des fonctions permettant d'accéder au premier, au dernier élément, aux i-èmes éléments à partir du début ou de la fin de la structure.
4. Donnez une fonction vérifiant si les éléments de σ_1 et σ_2 sont égales. Est-ce qu'alors σ_1 et σ_2 sont égales ?
5. Donnez une fonction supprimant un élément d'une telle structure.

Matthieu Boyer
Planche B

Question de Cours

Rappelez la différence entre une liste de Python et un tableau de C.

Vectors

On écrira le code de cet exercice en OCaml. On donnera toujours la complexité des différentes fonctions écrites.
Vous avez vu en cours une implémentation des tableaux dynamiques en C, nous allons la réétudier en OCaml.

1. Proposez une implémentation de tableaux dynamiques/redimensionnables.
2. Donnez une fonction permettant d'accéder au k ème élément d'un tableau dynamique.
3. Donnez des fonctions permettant de créer un tableau dynamique à partir d'une liste et d'un tableau.
4. Donnez une fonction permettant d'ajouter et une permettant d'enlever un élément d'un tableau dynamique. Ici, donner la complexité n'a que peu de sens, donnez la complexité amortie de l'opération.

La Paresse

On se propose d'implémenter en OCamL une bibliothèque de calcul paresseux. Celle-ci expose un type paramétré de suspensions `'a susp` et des fonctions de types suivants :

— `susp: (unit -> 'a) -> 'a susp`

— `force: 'a susp -> 'a`

Une suspension (de type `'a susp`) contient une fonction permettant de calculer une valeur de type `'a`. Elle peut être construite facilement grâce à la fonction `susp`. Le calcul n'est effectué que lorsque l'utilisateur de la bibliothèque le demande via la fonction `force`. Cette dernière fonction vérifie si le calcul a déjà été effectué : si tel est le cas, elle en renvoie le résultat pré-calculé. Sinon, elle lance le calcul, stocke le résultat pour de futurs appels, et elle le renvoie.

1. Donner une implémentation possible de cette bibliothèque, dans le langage OCamL. On pourra remarquer qu'une fonction n'est pas à priori constante et donc évaluée.

On définit le type des *listes paresseuses* en OCamL :

```
type 'a slist_cell =
| SNil
| SCons of 'a * 'a slist
and 'a slist = 'a slist_cell susp;;
```

2. (a) Comparer la notion de liste paresseuse avec la notion habituelle de liste.
   (b) Écrire une fonction `scons: 'a -> 'a slist -> 'a slist` qui ajoute un élément en tête d'une liste paresseurse, ainsi qu'une valeur `snil` de liste paresseuse vide.
   (c) Écrire deux fonctions `shd: 'a slist -> 'a` et `stl: 'a slist -> 'a slist` qui prennent une liste paresseuse non vide en paramètre et qui renvoie respectivement son premier élément et la liste paresseuse des autres éléments.
   (d) Écrire une fonction `sappend: 'a slist -> 'a slist -> 'a slist` qui concatène de manière paresseuse deux listes paresseuses. Cette fonction devra s'exécuter en temps constant.
   (e) Écrire une fonction `srev: 'a slist -> 'a slist` qui renverse une liste paresseuse de manière efficace. Quelle est la complexité des accès aux différents éléments de la liste renversée ?

Pointeurs sur un modèle de RAM

On donne la représentation suivante d'une mémoire vive :

```
type bit = Zero | One;;
ntype address = bit array;;
ntype ram = bit array;;
```

Ici, on représente la RAM comme un tableau binaire. On stockera le pointeur vers le début de la stack sur le premier octet de la RAM. On supposera que tous nos entiers se représentent sur 8 bits en petit-boutiste.

1. Donnez une fonction `init: ram` qui crée une RAM vide.
2. Donnez une fonction `référencer: int -> ram -> address` qui a un entier et une ram stocke l'entier dans la ram et renvoie un pointeur vers l'entier. On agira sur la ram avec effet de bord.
3. Donnez une fonction `modifier: address -> int -> ram -> unit` qui a un pointeur et un entier modifie la valeur de l'entier associée au pointeur.
4. Donnez une fonction `déréférencer: ram -> address -> int` qui a une ram et un pointeur et renvoie la valeur de l'entier associé au pointeur.
5. Définissez une fonction `arith: address -> int -> address` telle que `arith p n` renvoie le pointeur vers l'adresse définie par $p + n$.
6. Donnez une fonction `free: address -> ram -> unit` qui libère la variable associée au pointeur.