

Logique, Dédution, Programmation - Master 1 d'informatique

Planche de TD 2 - Les Listes

2009-2010

I- Programmer en Caml les fonctions suivantes (dont on donnera le type dans chaque cas):

- (a)- *last* qui renvoie le dernier élément d'une liste.
- (b)- n^{th} , qui étant donné un entier n et une liste l , renvoie le n^{ieme} élément de l s'il existe. On considère que la tête a pour numéro 0.
- (c)- *carré* qui renvoie la liste des carrés d'une liste d'entiers.
- (d)- *succ*, qui renvoie la liste des successeurs d'un liste d'entiers.
- (e)- *map* qui étant donnée une fonction f et une liste l , renvoie la liste des images par f des éléments de l . Elle vérifie:

$$map\ f\ [a_1; \dots; a_n] = (f\ a_1; \dots; f\ a_n)$$

Redéfinir *carré* et *succ* à l'aide de *map*.

- (f)- *prodlis*, qui renvoie la liste formée des produits terme à terme de 2 listes d'entiers de même longueur. Elle est telle que:

$$prodlis\ [a_1; \dots; a_n]\ [b_1; \dots; b_n] = [a_1 * b_1; \dots; a_n * b_n].$$

- (g)- *map₂* telle que :

$$map_2\ f\ [a_1; \dots; a_n]\ [b_1; \dots; b_n] = [f\ a_1\ b_1; \dots; f\ a_n\ b_n]$$

- (h)- Redéfinir *prodlis* à l'aide de *map₂*.

On rappelle qu'il existe deux itérateurs sur les listes, prédéfinis en Caml. Ils sont tels que:

$$\begin{aligned} (fold_left\ f\ a\ [b_1; \dots; b_n]) &= (f\ \dots\ (f\ (f\ a\ b_1)\ b_2)\ \dots\ b_n) \\ (fold_right\ f\ [a_1; \dots; a_n]\ b) &= (f\ a_1\ (f\ a_2\ (\dots\ (f\ a_n\ b)\ \dots))) \end{aligned}$$

II- Définir les fonctions calculant la longueur d'une liste, son inverse, la fonction *map*, la concaténation de deux listes en utilisant *fold.left* ou *fold.right*.

III- Programmer les fonctions suivantes (on utilisera les itérateurs chaque fois que l'algorithme suppose un parcours de toute la liste passée en argument):

- (a)- *mem* telle que (*mem* $a\ l$) vaut *true* si et seulement a est élément d'une liste l .
- (b)- *except* telle que (*except* $a\ l$) renvoie l privée du premier élément égal à a , laisse l inchangée sinon.
- (c)- *subtract* telle que (*subtract* $l_1\ l_2$) renvoie la liste des éléments de l_1 qui ne sont pas dans l_2 .
- (d)- *union* telle que (*union* $l_1\ l_2$) concatène les éléments de l_1 qui ne sont pas dans l_2 aux éléments de l_2 .