Master Parisien de Recherche en Informatique 2005–2006

**Examen cours 2-7-2 Assistants de preuve**

Jeudi 23 février 2006

L'énoncé est composé de 4 pages. L'examen dure 3 heures. Les notes de cours manuscrites ainsi que les supports de cours distribués cette année sont les seuls documents autorisés.

# De l'exponentiation rapide aux matrices carrées : une aventure au pays des types

## 1 Exponentiation rapide

On rappelle l'algorithme d'exponentiation rapide permettant de calculer $x^n$ en $O(\log n)$ multiplications :

```
let rec fastexp c x n = (* calcule c * x^n *)
  if n = 0 then c
  else if n mod 2 = 0 then fastexp c (x*x) (n/2)
  else fastexp (c*x) (x*x) (n/2)
let exp x n = fastexp 1 x n
```

Le but de cette première partie est simplement de définir cette fonction en Coq, à titre d'échauffement.

1. Définir une fonction `half : nat → nat` telle que `half` $n = \lfloor \frac{n}{2} \rfloor$ par récurrence structurelle sur son argument.
2. Définir une fonction `even : nat → bool` caractérisant les entiers pairs par récurrence structurelle sur son argument.
3. On va définir la fonction `fastexp` par récurrence structurelle sur la preuve d'un prédicat ad-hoc.
   (a) Définir un prédicat inductif `half_dom : nat → Prop` dans ce but.
   (b) Énoncer le ou les lemmes d'inversions nécessaires. (On ne cherchera pas à en donner les preuves.)
   (c) Définir une fonction `fastexp : nat → nat → ∀n:nat, half_dom n → nat`, telle que `fastexp` $c\ x\ n\ h = c \times x^n$, par récurrence structurelle sur la preuve de `half_dom n`.
4. En déduire une fonction `exp : nat → ∀n:nat, half_dom n → nat` telle que `exp` $x\ n\ h = x^n$.
5. Définir un terme clos `half_dom_5` de type `half_dom 5`. Indiquer si la commande `Eval compute in exp 2 5 half_dom_5` doit aboutir ou non sur la valeur 32.

## 2 Vecteurs

On s'inspire maintenant de l'algorithme d'exponentiation rapide ci-dessus pour définir un type Coq `vector A` pour des vecteurs d'éléments de type `A` dans lesquels la longueur est encodée dans la représentation, permettant ainsi des opérations efficaces. On commence par la définition d'un type `vector_` qui suit la fonction `fastexp` :

```
Inductive vector_ (v w : Set) : Set :=
  | Vzero : v -> vector_ v w
  | Veven : vector_   v   (w*w) -> vector_ v w
  | Vodd  : vector_ (v*w) (w*w) -> vector_ v w.
```

puis on définit le type `vector` en utilisant le type `unit` (type prédéfini de sorte `Set` possèdant une unique valeur, `tt`) en lieu et place de la constante 1 :

```
Definition vector A := vector_ unit A.
```

Ainsi le vecteur à 5 éléments $(a, b, c, d, e)$ sera représenté par la valeur

```
Vodd (Veven (Vodd (Vzero ((tt, a), ((b,c), (d,e))))))
```

(Les types arguments des constructeurs ont été volontairement omis.)

1. Définir une fonction `vdim : ∀A:Set, vector A → nat` calculant la dimension d'un vecteur. Indication : commencer par définir une fonction

   ```
   vdim_ (A B: Set) (nv nw: nat) (v: vector_ A B) : nat
   ```

   par récurrence structurelle sur `v`, `nv` et `nw` étant les « dimensions » respectives des types `A` et `B`.

2. Définir une fonction

   ```
   vcreate (A: Set) (a: A) (n: nat) (h: half_dom n) : vector A
   ```

   créant un vecteur de taille `n` dont tous les éléments sont `a`. Indication : commencer par définir une fonction

   ```
   vcreate_ (A B:Set) (v:A) (w:B) (n:nat) (h:half_dom n) : vector_ A B
   ```

   procédant par récurrence structurelle sur la preuve `h`.

3. Définir une fonction

   ```
   vget (A: Set) (i: nat) (v: vector A) : option A
   ```

   accédant au `i`-ième élément d'un vecteur `v` i.e. telle que `vget` $A\ i\ v$ = `Some` $v_i$ lorsque $v$ représente le vecteur $(v_0, v_1, \ldots, v_{n-1})$. Le comportement pour une valeur de `i` supérieure ou égale à $n$ n'est pas spécifié. Là encore, on commencera par définir une fonction `vget_` opérant sur le type `vector_`. On pourra utiliser la fonction `le_lt_dec` : $\forall n\, m$, `{`$n \leq m$`}+{`$m < n$`}` de la bibliothèque standard de Coq.

4. Quelle est la complexité de cette fonction d'accès (en fonction de la taille $n$ du vecteur) ?

# 3 Matrices carrées

Une matrice peut être définie comme un vecteur de vecteurs. Cependant, si l'on souhaite définir des matrices carrées, le typage ne permettra pas de garantir qu'une telle matrice est *effectivement* carrée, c'est-à-dire que son nombre de lignes est égal à son nombre de colonnes.

Pour y remédier, on transpose l'idée précédente au niveau des *opérateurs de types* c'est-à-dire de `Set` vers `Set` $\rightarrow$ `Set`. Dans ce nouvel univers, le produit, l'élément neutre et l'identité sont définis par

```
Definition Prod (v w: Set -> Set) (a: Set) := ((v a)*(w a))%type.
Definition Empty (a:Set) : Set := unit.
Definition Id (a:Set) : Set := a.
```

On définit alors le type `square A` des matrices carrées d'éléments de type `A` de la manière suivante :

```
Inductive square_ (v w: Set -> Set) (A: Set) : Set :=
  | Mzero : v (v A)                          -> square_ v w A
  | Meven : square_ v          (Prod w w) A -> square_ v w A
  | Modd  : square_ (Prod v w) (Prod w w) A -> square_ v w A.

Definition square : Set -> Set := square_ Empty Id.
```

1. Donner un terme `m_3_3` de type `square nat` représentant la matrice $3 \times 3$

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix}$$

2. Définir une fonction `mdim : ∀A:Set, square A → nat` calculant la dimension d'une matrice. Comme pour les vecteurs, on commencera par une fonction

   ```
   mdim_ (v w:Set -> Set) (a:Set) (nv nw:nat) (m:square_ v w a) : nat
   ```

   opérant sur le type `square_` et par récurrence structurelle sur `m`, `nv` et `nw` étant les « dimensions » respectives des types `v a` et `w a`.

3. On souhaite définir une fonction

   ```
   mcreate (A:Set) (a:A) (n:nat) (h:half_dom n) : square A
   ```

   créant une matrice `n` $\times$ `n` dont tous les éléments sont `a`.

   (a) Définir une fonction

   ```
   mkP (v w: Set -> Set)
       (mkv: forall (b:Set), b -> v b)
       (mkw: forall (b:Set), b -> w b)
       : forall (b:Set), b -> Prod v w b
   ```

   combinant deux fonctions de construction `mkv` et `mkw` pour les opérateurs de types `v` et `w` en une fonction de construction pour l'opérateur `Prod v w`.

(b) Définir une fonction

```
mcreate_
  (v w: Set -> Set)
  (mkv: forall (b:Set), b -> v b)
  (mkw: forall (b:Set), b -> w b)
  (A:Set) (a:A) (n:nat) (h:half_dom n) : square_ v w A
```

construisant une matrice de type `square_ v w A` dont tous les éléments sont `a`, par récurrence structurelle sur la preuve `h`, à partir de deux fonctions de constructions `mkv` et `mkw` pour les opérateurs de types `v` et `w`.

(c) En déduire la définition de la fonction `mcreate`.

4. Définir une fonction

```
mget (A: Set) (i j: nat) (m: square A) : option A
```

accédant à l'élément $(\mathtt{i}, \mathtt{j})$ d'une matrice `m` de taille $n \times n$, `i` étant l'indice de la ligne et `j` celui de la colonne, tous deux compris entre 0 et $n-1$. Pour une valeur de `i` ou `j` supérieure ou égale à $n$, le comportement n'est pas spécifié. Là encore, on commencera par définir une fonction `mget_` opérant sur le type `square_`.