FSAB1402: Informatique 2

Programmer avec l'Etat

Peter Van Roy
Département d'Ingénierie Informatique, UCL

pvr@info.ucl.ac.be

Ce qu'on va voir aujourd'hui

- Les questions de l'interrogation et leur réponse
- Quelques structures de données importantes avec et sans état
  - Tuple et enregistrement (sans état)
  - Tableau et dictionnaire (avec état)
- Une comparaison des deux modèles (déclaratif et avec état)
  - En définissant des opérations sur des matrices

Les questions de l'interro

P. Van Roy

Question 1

- Ecrivez une fonction déclarative Strip qui prend une liste d'entiers Xs=[x_1, x_2,…, x_n] et qui renvoie la liste Ys=[x_2, x_3,…, x_{n-1}] qui est la liste Xs sans le premier et le dernier élément
- Chaque fonction récursive doit être une boucle!

P. Van Roy

Question 2: Rechercher des informations

- Supposez que l'arbre T est dégénéré en forme de liste. Calculez la complexité temporelle.
- Supposez que l'arbre T est équilibré
  - Démontrez que le nombre de nœuds est 2^k-1
  - Si $T_1(k)$ est le temps d'exécution pour un arbre équilibré avec 2^k-1 nœuds, démontrez que $T_1(k) \leq T_1(k-1)+c$
  - Et $T_2(n)$, le temps d'exécution pour un arbre de n nœuds?

```
fun {Lookup X T}
   case T
   of leaf then notfound
   [] tree(key:Y value:V T1 T2) andthen X==Y then
         found(V)
   [] tree(key:Y value:V T1 T2) andthen X<Y then
         {Lookup X T1}
   [] tree(key:Y value:V T1 T2) andthen X>Y then
         {Lookup X T2}
   end
end
```


Question 3 (1)

Voici un fragment de programme:

```
local X Y P in
      Y=10
      proc {P}
            X=Y+1
      end
      local Y in
            Y=20
            {P}
            {Browse X}
      end
end
```

- Qu'est-ce qui est affiché quand ce fragment est exécuté?
- Quel est l'environnement contextuel de la procédure P?
- Quel est le contenu de la mémoire après l'exécution du fragment?

Question 3 (2)

- Définissez avec précision:
  - Identificateur
  - Variable en mémoire
  - Environnement
  - Environnement contextuel

Résumé du dernier cours

La sémantique

- Il est important de comprendre comment exécute un programme
  - Celui qui ne comprend pas quelque chose est l'esclave de cette chose
- Il faut pouvoir montrer l'exécution d'un programme selon la machine abstraite
  - Concepts importants: environnement contextuel ("le lieu de naissance"), pile sémantique ("ce qu'il reste à faire"), définition et appel de procédure
- Pour les exercices: il ne faut pas sombrer dans les détails
  - Il suffit de les donner une fois, après vous pouvez faire des raccourcis (comme par exemple, sauter des pas, ne montrer qu'une partie de la pile, de la mémoire et des environnements, utiliser des substitutions)

L'état

- L'état explicite (la cellule)
- L'avantage pour la modularité des programmes
  - Etendre une partie sans devoir changer le reste
- La sémantique des cellules
  - Une cellule est une paire: le nom de la cellule et son contenu
  - Le nom de la cellule est aussi appelé l'adresse
  - Les cellules habitent la mémoire à affectation multiple

L’abstraction de données

- Motivations
  - Donner des garanties
  - Réduire la complexité
  - Faire de grands programmes en équipe
- Les deux formes principales
  - Le type abstrait et l’objet
- L’utilité de chaque forme et la mise en oeuvre en Java
- Le type abstrait avec état

Collections indexées

Collections indexées

- Une collection indexée regroupe un ensemble de valeurs
- Chaque élément est accessible par l'indexe
- Dans le modèle déclaratif il y a deux types de collections indexées:
  - Les tuples, par exemple date(17 mars 2005)
  - Les enregistrements, par exemple date(jour:17 mois:mars annee:2005)
- Avec l'état on peut définir d'autres types de collections:
  - Tableaux ("arrays")
  - Dictionnaires

Tableaux ("arrays")

- Un tableau est une correspondance entre entiers et valeurs
  - C'est-à-dire, un ensemble de valeurs indexé par des entiers
- Le domaine du tableau est un ensemble d'entiers consécutifs, avec une borne inférieure et une borne supérieure
  - Le domaine ne peut pas être changé
  - Le contenu (les éléments) peut être changé
- On peut considérer un tableau comme un tuple de cellules

Opérations sur les tableaux

- A={Array.new LB UB V}
  - Créé un tableau A avec borne inférieure LB et borne supérieure UB
  - Tous les éléments sont initialisés a V
- Les autres opérations
  - Accès et mise à jour des éléments
  - Obtenir les bornes
  - Convertir un tableau en tuple et inversément
  - Tester le type d'un tableau

Exemple

- A={MakeArray L H F}
- Créé un tableau A où chaque élément I a la valeur {F I}

```
fun {MakeArray L H F}
   A={Array.new L H 0}
in
   for I in L..H do
      A.I := {F I}
   end
   A
end
```

Convertir un tuple en tableau

```
fun {Tuple2Array T}
   H={Width T}
in
   {MakeArray
    1 H
    fun {$ I} T.I end}
end
```

P. Van Roy

Convertir un tableau en enregistrement

- R={Array2Record L A}
  - Prend une étiquette L et un tableau A, renvoie un enregistrement R don't l'étiquette est L et dont les noms des champs sont des entiers de la borne inférieure jusqu'à la borne supérieure de A
- Pour définir cette fonction, nous devons savoir comment construire un enregistrement
- R={Record.make L Fs}
  - Construit un enregistrement R avec étiquette L et une liste de noms de champs Fs, et le contenu des champs sont des variables libres
- L={Array.low A} et H={Array.high A}
  - Renvoyer les bornes inférieure et supérieure de A

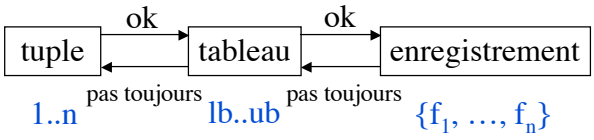
P. Van Roy

Définition de Array2Record

```
fun {Array2Record LA A}
   L={Array.low A}
   H={Array.high A}
   R={Record.make LA {From L H}}
in
   for I in L..H do
      R.I = A.I
   end
   R
end
```

Attention! Ceci n'est pas une affectation de cellule (":="), mais une affectation de variable ("="). Affectation unique alors!

Conversions entre collections



- On peut convertir n'importe quel tuple en un tableau
- Mais on ne peut pas convertir n'importe quel tableau en un tuple
  - Pourquoi?
- On peut convertir n'importe quel tableau en un enregistrement
- Une conversion de tableau en tuple ou en enregistrement est une "photographie instantanée"
  - Pourquoi on dit ça?

Dictionnaires (tables de hachage)

- Un dictionnaire est une correspondance entre valeurs simples (des litéraux: entiers et atomes) et des valeurs quelconques
  - C'est-à-dire, un ensemble de valeurs indexé par des litéraux
- La paire (litéral, valeur) s'appelle un item
  - Le litéral s'appelle la clé
- Le domaine peut être changé
  - On peut ajouter de nouveaux items et enlever des items
  - Le temps pour ces opérations est un temps constant amorti
  - C'est-à-dire, *n* opérations prennent un temps *O(n)*

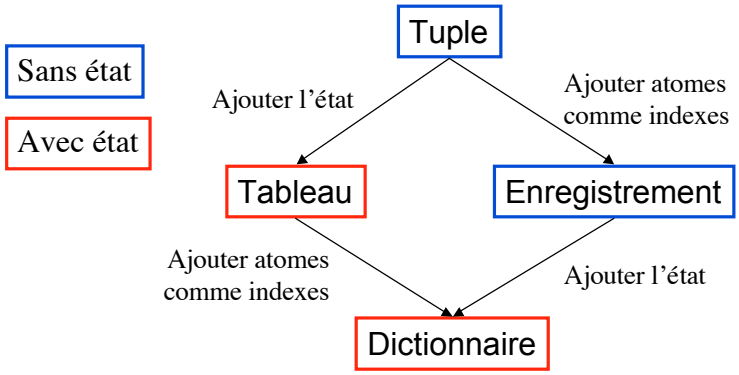
Opérations sur les dictionnaires

- D={Dictionary.new}
  - Créé un nouveau dictionnaire vide
- Les autres opérations
  - Accès et mise à jour des éléments
  - Ajout et enlèvement d'un item
  - Tester si une clé est dans le dictionnaire
  - Convertir un dictionnaire en enregistrement et inversément
  - Tester le type d'un dictionnaire

Implémentation des dictionnaires

- L'accès à un élément se fait en un temps constant
- Les opérations d'ajout et d'enlèvement se font en un temps constant *amorti*
- Qu'est-ce que cela veut dire exactement?
  - *n* opérations se font en un temps *O(n)*
- Pourquoi l'ajout et l'enlèvement ne se font pas tout bêtement en temps constant?
  - L'espace mémoire utilisé par un dictionnaire est proportionnel au nombre d'éléments
  - Pour garder un temps constant d'accès, le dictionnaire est organisé comme une table de hachage
  - Quand on ajoute ou enlève un élément, il faut parfois reorganiser cette table pour garantir le temps constant d'accès

Hierarchie des collections indexées



- Voici un diagramme qui montre les relations entre les différents types de collections indexées

D’autres collections

- Collections déclaratives
  - Listes
  - Flots (listes sans fin)
  - Piles en type abstrait
  - Files en type abstrait
- Collections avec état
  - Piles en objet
  - Files en objet

Matrices

Comparaison: déclaratif versus état

- Pour compléter la comparaison des deux modèles, déclaratif et avec état, nous allons regarder deux implémentations de quelques algorithmes sur les matrices
- Une première série d'implémentations utilisera un type déclaratif et sera déclarative
- Une seconde série d'implémentations utilisera un type avec état et sera avec état
- Nous allons commencer par donner une description abstraite des opérations à implémenter indépendante de tout modèle
- Quel modèle est le meilleur? Nous allons voir!

P. Van Roy

Une matrice

- Une matrice A est un ensemble A=[A_{ij}] de *mxn* éléments organisé en un rectangle avec *m* rangées et *n* colonnes:

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{pmatrix}$$

P. Van Roy

Opérations sur les matrices

- Les matrices sont beaucoup utilisées dans différents domaines
- Aujourd'hui, nous allons définir deux opérations sur les matrices, l'addition et la multiplication
- Nous allons définir chaque opération dans le modèle déclaratif et dans le modèle avec état, pour comparer les deux modèles

Addition des matrices

- Voici la définition de l'addition de deux matrices $[A_{ij}]$ et $[B_{ij}]$ de taille mxn:
  - $[A_{ij}]+[B_{ij}] = [A_{ij}+B_{ij}]$
- Pour implémenter cette définition, nous allons choisir deux représentations d'une matrice:
  - Représentation déclarative: une liste de listes [[A11 A12 … A1n] … [Am1 Am2 … Amn]]
  - Représentation avec état: un tableau dont les éléments sont des tableaux, l'élément A_{ij} est A.I.J

Addition déclarative

```
fun {AddM A B}
   case A#B of nil#nil then nil
   [] (AR|A2)#(BR|B2) then
         {AddRow AR BR}|{AddM A2 B2}
   end
end
fun {AddRow AR BR}
   case AR#BR of nil#nil then nil
   [] (AE|AR2)#(BE|BR2) then
         (AE+BE)|{AddRow AR2 BR2}
   end
end
```

P. Van Roy

Addition avec état

```
fun {AddM A B}
   M={Array.high A}
   N={Array.high A.1}
   C={Array.new 1 M 0}
in
   for I in 1..M do
         C.I:={Array.new 1 N 0}
         for J in 1..N do
               C.I.J:=A.I.J+B.I.J
         end
   end
   C
end
```

P. Van Roy

Comparaison des deux définitions

- Les définitions ont une complexité comparable
  - Le temps et l'espace d'exécution sont comparables aussi
  - La définition déclarative est néanmoins plus difficile à lire, pourquoi?
- Dans la définition déclarative, chaque boucle est une fonction récursive. Deux boucles imbriquées deviennent deux fonctions récursives, dont la première appelle la seconde.
- Dans la définition avec état, il faut plus d'effort pour initialiser les structures, avec des appels à Array.high et Array.new

Multiplication des matrices

- Voici la définition de la multiplication de deux matrices $[A_{ij}]$ et $[B_{ij}]$ de taille mxp et pxn:
  - $[A_{ij}]x[B_{ij}] = [\Sigma_k A_{ik}B_{kj}]$
- Cette fois nous aurons besoin de trois boucles imbriquées: deux pour les rangées et les colonnes, et une pour la somme intérieure
- Pour implémenter cette définition, nous allons choisir deux représentations d'une matrice:
  - Deuxième représentation déclarative: un tuple dont les éléments sont des tuples, l'élément A_{ij} est A.I.J
  - Représentation avec état: un tableau dont les éléments sont des tableaux, l'élément A_{ij} est A.I.J

Multiplication déclarative (1)

```
fun {MulM A B}
   M={Width A} P={Width A.1}  N={Width B.1}
   C={Tuple.make m M}
in
   for I in 1..M do
      C.I={Tuple.make r N}
      for J in 1..N do
         C.I.J = (Somme de (A.I.K*B.K.J) pour K=1..P)
      end
   end
   C
end
```


Multiplication déclarative (2)

```
fun {MulM A B}
   M={Width A} P={Width A.1}  N={Width B.1}
   C={Tuple.make m M}
in
   for I in 1..M do
      C.I={Tuple.make r N}
      for J in 1..N do
         fun {Sum K Acc}
            if K>P then Acc else {Sum K+1 (A.I.K*B.K.J)+Acc} end
         end
      in
         C.I.J={Sum 1 0}
      end
   end
   C
end
```

Multiplication avec état

```
fun {MulM A B}
   M={Array.high A} P={Array.high A.1} N={Array.high B}
   C={Array.new 1 M 0}
in
   for I in 1..M do
         C.I:={Array.new 1 N 0}
         for J in 1..N do
            for K in 1..P do
                  C.I.J:=(A.I.K*B.K.J)+C.I.J
            end
         end
   end
   C
end
```


Comparaison des deux définitions

- Les définitions ont une complexité comparable
  - Le temps et l'espace d'exécution sont comparables aussi
  - La définition de Sum utilise un accumulateur: c'est un peu plus compliqué
- Dans la définition déclarative, il faut faire attention à n'affecter chaque élément du tuple qu'une seule fois
  - C'est pourquoi il faut parfois des définitions récursives (comme la définition de Sum avec son accumulateur)
- Si le programme est concurrent (il y a un autre programme qui utilise $[C_{ij}]$ en même temps qu'il est calculé), la définition déclarative marchera sans changements. La définition avec état devrait être changée (utilisation des verrouillages).

Exercice

- Remarquez qu'on n'a plus utilisé la première représentation déclarative (liste des listes) pour la multiplication
  - On a préféré deux représentations plus ou moins semblables: tuple de tuples et tableau de tableaux
- Comme exercice, je vous conseille d'écrire une définition déclarative avec la première représentation déclarative (liste des listes)
  - C'est nettement plus compliqué parce qu'une liste ne permet pas un accès immédiat à n'importe quel élément. Il faut manipuler les listes pour qu'on puisse faire les calculs en traversant les listes du début à la fin.

Exercice (tuyau: partie 1)

```
fun {MulM A BT}
   case A of nil then nil
   [] AR|A2 then {MulRowM AR BT}|{MulM A2 BT}
   end
end
fun {MulRowM AR BT}
   case BT of nil then nil
   [] BC|BT2 then {RowColM AR BC 0}|{MulRowM AR BT2}
   end
end
fun {RowColM AR BC Acc}
   case AR#BC of nil#nil then Acc
   [] (A|AR2)#(B|BC2) then {RowColM AR2 BC2 Acc+A*B}
   end
end
```

Exercice (tuyau: partie 2)

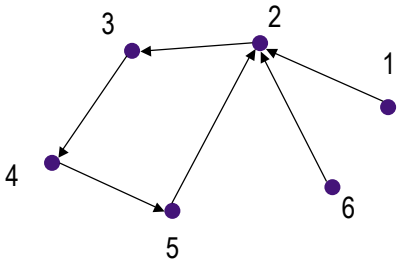
- Il faut la transposition de B, qu'on note comme BT, comme argument à MulM
- Il suffit alors de définir une fonction qui fait la transposition d'une matrice
  - **fun** {TransM B} --> BT
- Pour définir TransM il faut deux fonctions récursives parce qu'il y a deux boucles imbriquées
  - Exercice!
- Après il faut tester votre définition complète pour vérifier qu'elle marche comme prévu

Un autre exemple

Un autre exemple: la fermeture transitive

- Le livre du cours donne un autre exemple pour comparer les deux modèles (voir la section 6.8.1)
- L'algorithme choisi est la fermeture transitive sur les graphes orientés
  - Un graphe orienté est un ensemble de noeuds et des arêtes entre les noeuds
- Je vous demande de lire cette section vous-mêmes et de la comprendre

Fermeture transitive d'un graphe



Les noeuds: {1,2,3,4,5,6}

Les arêtes: { (1,2), (2,3), (3,4), (4,5), (5,6), (6,2), (1,2) }

- Fermeture transitive: à partir d'un graphe G, calculer un autre graphe T, avec les mêmes noeuds mais d'autres arêtes
- S'il y a un chemin entre deux noeuds en G, alors il y a un arête entre les deux noeuds en T

Résumé

Résumé

- Collections indexées
  - Tuple
  - Enregistrement
  - Tableau (avec état, indexes sont des entiers)
  - Dictionnaire (avec état, indexes sont des litéraux)
- Matrices
  - Addition et multiplication
  - Dans les modèles déclaratifs et avec état
  - Comparaison des modèles