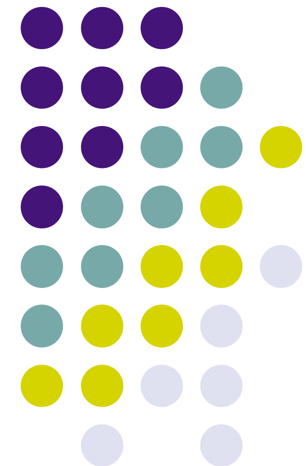
FSAB1402: Informatique 2
Programmer avec des Types Abstraits



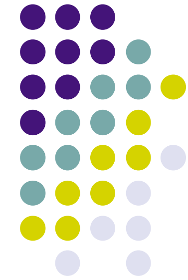
Peter Van Roy

Département d'Ingénierie Informatique, UCL

pvr@info.ucl.ac.be

© 2007 P. Van Roy. All rights reserved.

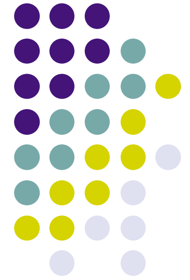
Ce qu'on va voir aujourd'hui



- Programmer avec des types abstraits
- Quelques abstractions importantes avec et sans état
  - Tuple et enregistrement (sans état)
  - Tableau et dictionnaire (avec état)
  - Ces structures seront données comme des types abstraits
- Ecrire des algorithmes avec les modèles déclaratifs et avec état
  - En définissant des opérations sur des matrices avec plusieurs représentations
  - (Autre exemple: fermeture transitive sur un graphe orienté)
- L'autre manière de faire une abstraction de données, l'objet, sera expliquée la semaine prochaine

© 2007 P. Van Roy. All rights reserved.

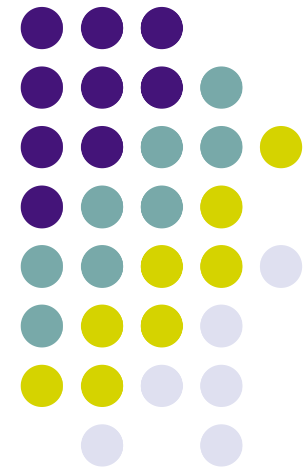
Lecture pour la huitième séance



- Chapitre 3 (section 3.5)
  - Les types abstraits
- Chapitre 5 (sections 5.4 et 5.7)
  - L'abstraction de données
  - Exercices!

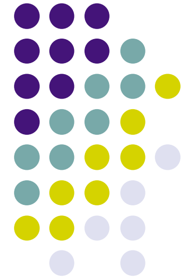
© 2007 P. Van Roy. All rights reserved.

Résumé
du dernier cours



© 2007 P. Van Roy. All rights reserved.

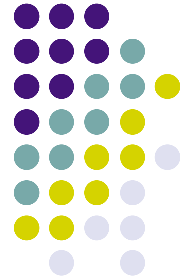
La sémantique



- Il est important de comprendre comment s'exécute un programme
  - Celui qui ne comprend pas quelque chose est l'esclave de cette chose
- Il faut pouvoir exécuter vous-mêmes un programme selon la machine abstraite
  - Concepts importants: environnement ("lien entre instruction et mémoire"), pile sémantique ("ce qu'il reste à faire"), définition et appel de procédure, environnement contextuel ("la valise d'une procédure")
- Pour les exercices: attention aux détails!
  - Il suffit de montrer tous les détails une fois; ensuite vous pouvez faire des raccourcis (comme par exemple, sauter des pas, utiliser des abréviations pour des environnements qui reviennent, etc.)

© 2007 P. Van Roy. All rights reserved.

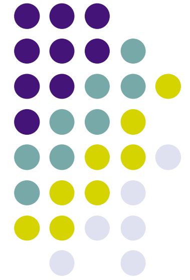
L’état



- L’état explicite (la cellule): un concept à double tranchant
  - Avantageux pour la modularité des programmes
    - Etendre une partie sans devoir changer le reste
  - Désavantageux pour l’exactitude des programmes
    - Un programme qui marche aujourd’hui peut être cassé demain
  - La solution pour avoir le meilleur des deux modèles: faire une grande partie du programme en modèle déclaratif avec des endroits isolés qui utilisent l’état
- La sémantique des cellules
  - Une mémoire à affectation multiple, qui contient des cellules
  - Une cellule est une paire: le nom et le contenu
  - Le nom de la cellule est aussi appelé l’adresse

© 2007 P. Van Roy. All rights reserved.

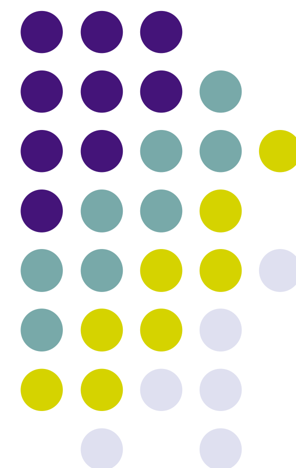
L’abstraction de données



- L’encapsulation et l’abstraction
  - L’encapsulation: protéger l’intérieur
  - L’abstraction: définir une interface pour une interaction contrôlée avec l’intérieur, ce qui garantit un bon comportement
- Motivations de l’abstraction de données
  - Donner des garanties
  - Réduire la complexité
  - Faire de grands programmes en équipe
- Les deux formes principales
  - Le type abstrait et l’objet

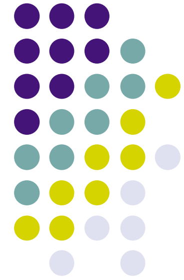
© 2007 P. Van Roy. All rights reserved.

Collections indexées



© 2007 P. Van Roy. All rights reserved.

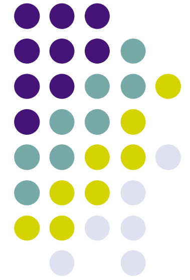
Collections indexées



- 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

© 2007 P. Van Roy. All rights reserved.

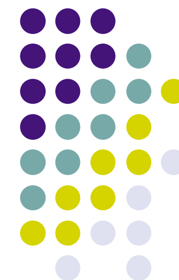
Tableaux (“arrays”)



- 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

© 2007 P. Van Roy. All rights reserved.

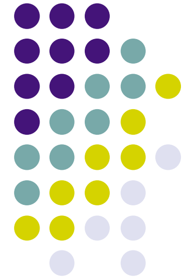
Opérations sur les tableaux



- 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

© 2007 P. Van Roy. All rights reserved.

Exemple

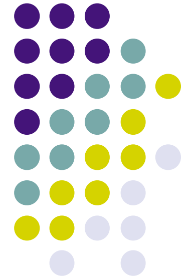


- A={MakeArray L H F}
- Créé un tableau A où chaque élément I a la valeur {F I}
- Remarquez que le tableau est un type abstrait en Oz

```
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
```

© 2007 P. Van Roy. All rights reserved.

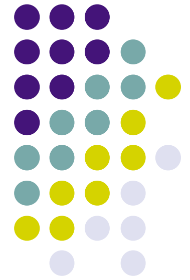
Convertir un tuple en tableau



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

© 2007 P. Van Roy. All rights reserved.

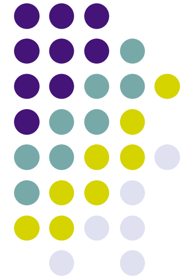
Convertir un tableau en enregistrement



- 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 les champs contiennent des variables libres
- L={Array.low A} et H={Array.high A}
  - Renvoyer les bornes inférieure et supérieure de A

© 2007 P. Van Roy. All rights reserved.

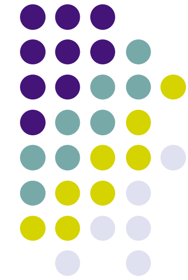
Définition de Array2Record



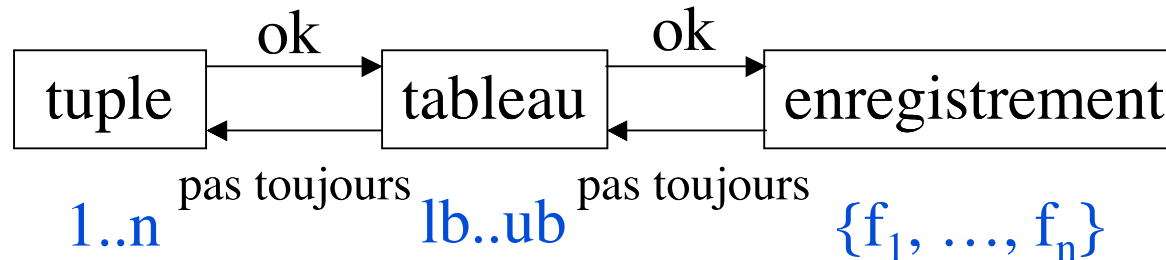
```
fun {Array2Record LA A}
   L={Array.low A}
   H={Array.high A}
   R={Record.make LA {From L H}}
in
   for I in L..H do
      R.I = A.I
   end
   R
end
```

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

© 2007 P. Van Roy. All rights reserved.



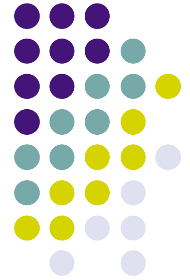
Conversions entre collections



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

© 2007 P. Van Roy. All rights reserved.

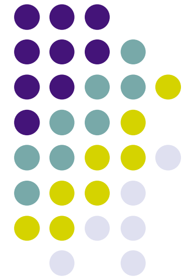
Dictionnaires (tables de hachage)



- Un dictionnaire est une correspondance entre valeurs simples (des littéraux: entiers ou atomes) et des valeurs quelconques
  - C’est-à-dire, un ensemble de valeurs indexé par des littéraux
- Une paire (littéral,valeur) s’appelle un item
  - Le littéral 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)*

© 2007 P. Van Roy. All rights reserved.

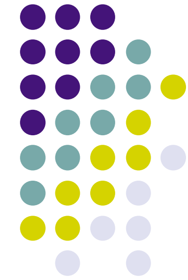
Opérations sur les dictionnaires



- 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
- Remarquez que le dictionnaire est un type abstrait en Oz

© 2007 P. Van Roy. All rights reserved.

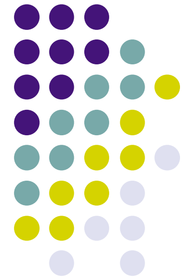
Implémentation des dictionnaires

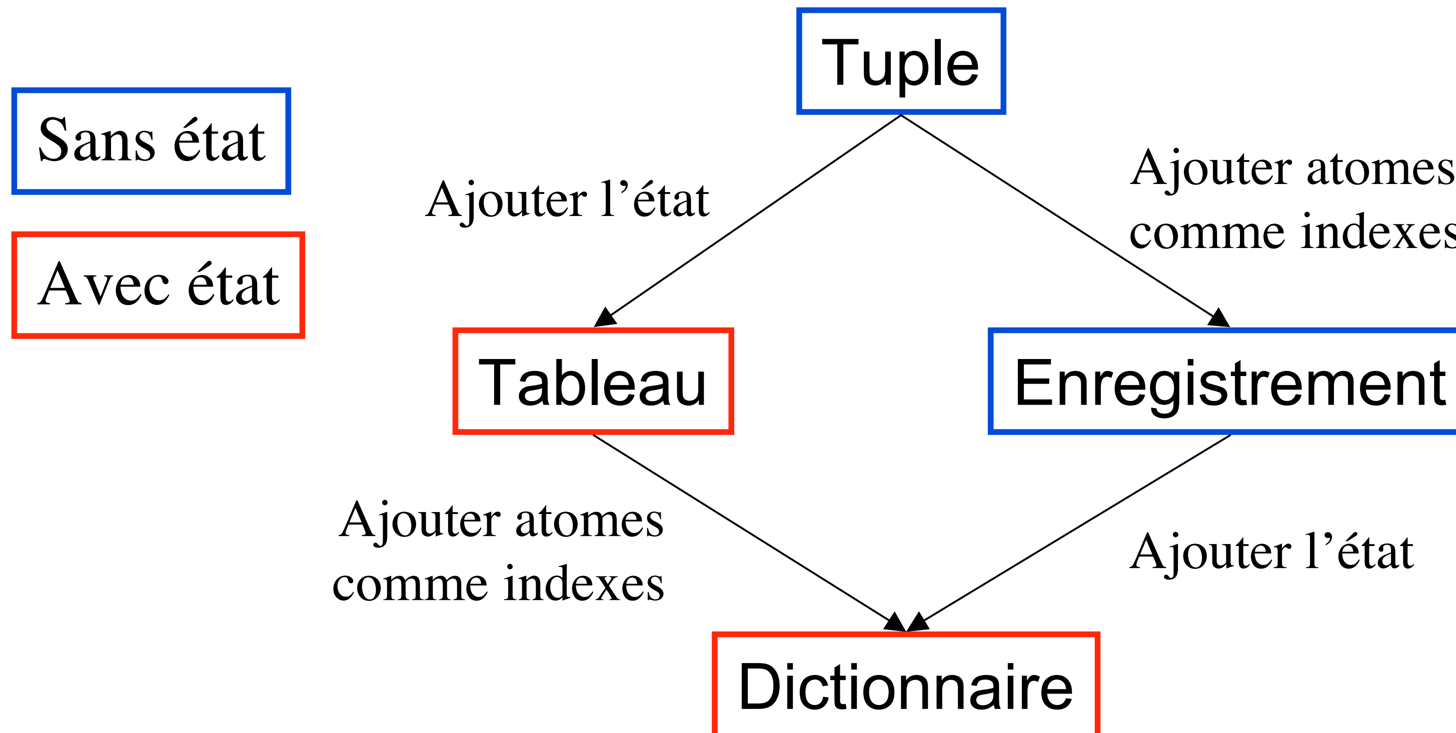


- L'accès à un élément se fait en un temps constant
- Les opérations d'ajout et d'enlèvement se font en un temps constant *amorti*
- Qu'est-ce que cela veut dire exactement?
  - *n* opérations se font en un temps *O(n)*
- Pourquoi l'ajout et l'enlèvement ne se font pas tout bêtement en temps constant?
  - 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

© 2007 P. Van Roy. All rights reserved.

Hierarchie des collections indexées

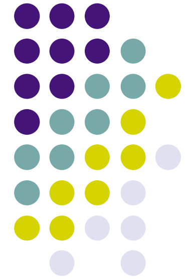




- Voici un diagramme qui montre les relations entre les différents types de collections indexées

© 2007 P. Van Roy. All rights reserved.

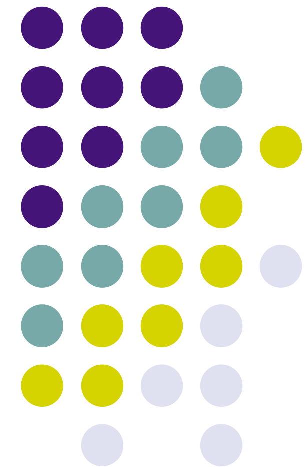
D’autres collections



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

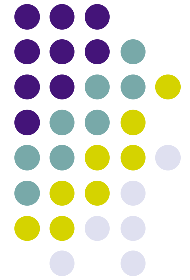
© 2007 P. Van Roy. All rights reserved.

Matrices



© 2007 P. Van Roy. All rights reserved.

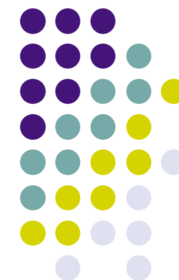
Comparaison des représentations des matrices



- Nous allons regarder des implémentations de quelques algorithmes sur les matrices
  - L'algorithme dépendra fortement de la représentation d'une matrice
  - Les représentations peuvent être déclaratives ou avec état
- Nous allons commencer par donner une description abstraite des opérations à implémenter indépendante de tout modèle
- Les matrices sont implémentées ici comme des types abstraits (valeurs + opérations)

© 2007 P. Van Roy. All rights reserved.

Une matrice

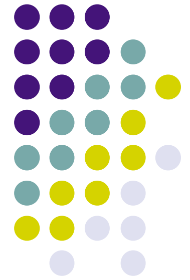


- Une matrice A est un ensemble $A=[A_{ij}]$ de *mxn* éléments organisé en un rectangle avec *m* rangées et *n* colonnes:

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

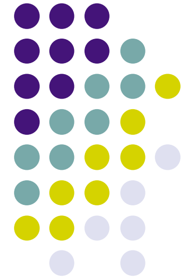
© 2007 P. Van Roy. All rights reserved.

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 avec plusieurs représentations
  - Une représentation peut être déclarative ou avec état
  - Attention, nos deux représentations seront toutes les deux des types abstraits!

© 2007 P. Van Roy. All rights reserved.

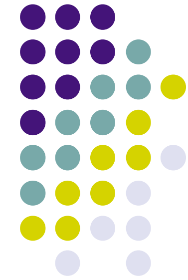


Addition des matrices

- 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 en liste: une liste de listes [[A11 A12 … A1n] … [Am1 Am2 … Amn]]
  - Représentation en tableau: un tableau dont les éléments sont des tableaux, l'élément A_{ij} est A.I.J

© 2007 P. Van Roy. All rights reserved.

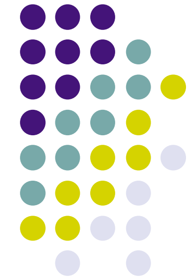
Addition pour la représentation en liste



```
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
```

© 2007 P. Van Roy. All rights reserved.

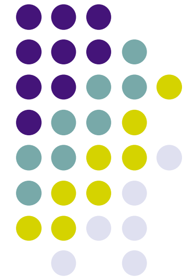
Addition pour la représentation en tableau



```
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
```

© 2007 P. Van Roy. All rights reserved.

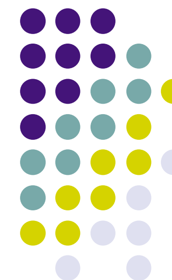
Comparaison des deux définitions



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

© 2007 P. Van Roy. All rights reserved.

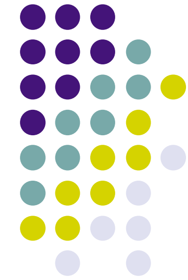
Multiplication des matrices



- 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:
  - Représentation en tuple (déclarative): un tuple dont les éléments sont des tuples, l'élément A_{ij} est A.I.J
  - Représentation en tableau (avec état): un tableau dont les éléments sont des tableaux, l'élément A_{ij} est A.I.J

© 2007 P. Van Roy. All rights reserved.

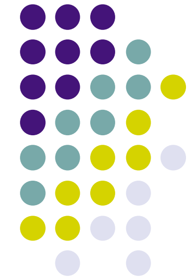
Multiplication pour la représentation en tuple (1)



```
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
```

© 2007 P. Van Roy. All rights reserved.

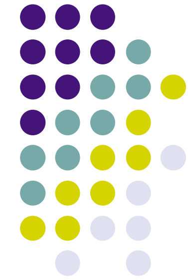
Multiplication pour la représentation en tuple (2)



```
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
```

© 2007 P. Van Roy. All rights reserved.

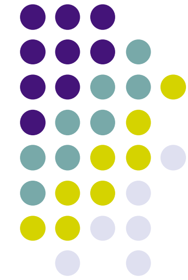
Multiplication pour la représentation en tableau



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

© 2007 P. Van Roy. All rights reserved.

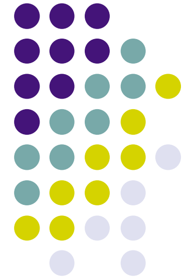
Comparaison des deux définitions



- 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).

© 2007 P. Van Roy. All rights reserved.

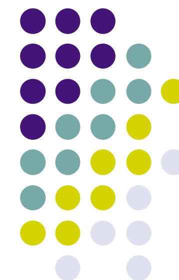
Exercice 1 (simple)



- Les types abstraits que nous avons donnés ne sont pas protégés
  - Les repésentations sont accessibles depuis l'extérieur des abstractions
- Etendez la définition de l'addition et la multiplication des matrices pour protéger la représentation interne
  - En utilisant Wrap et Unwrap, comme la définition de la pile la semaine dernière
  - Il faut créer Wrap et Unwrap avec NewWrapper

© 2007 P. Van Roy. All rights reserved.

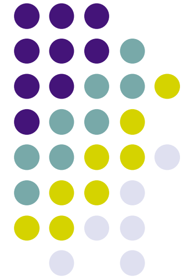
Exercice 2 (compliquée)



- Remarquez qu'on n'a plus utilisé la première représentation (liste des listes) pour la multiplication
  - On a préféré deux représentations plus ou moins semblables: tuple de tuples et tableau de tableaux
- Ecrivez une définition avec la première représentation (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.

© 2007 P. Van Roy. All rights reserved.

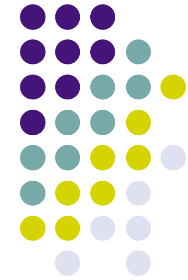
Exercice 2 (tuyau: partie 1)



```
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
```

© 2007 P. Van Roy. All rights reserved.

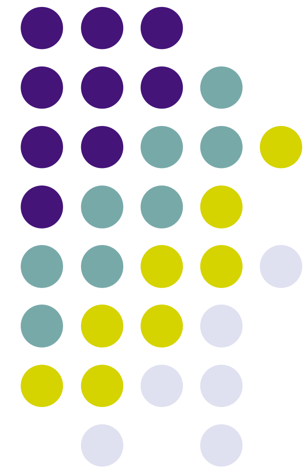
Exercice 2 (tuyau: partie 2)



- 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

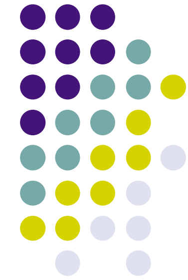
© 2007 P. Van Roy. All rights reserved.

Un autre exemple (supplément)



© 2007 P. Van Roy. All rights reserved.

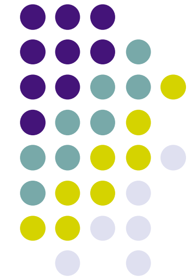
Un autre exemple: la fermeture transitive

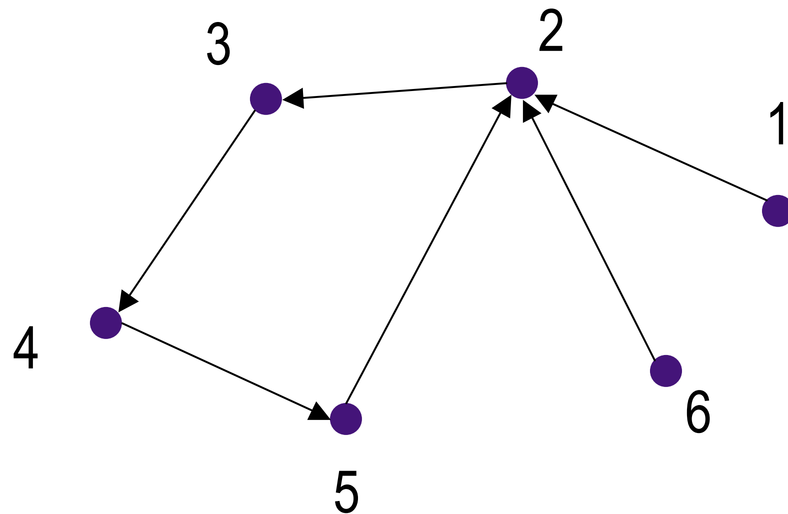


- Nous pouvons calculer la fermeture transitive d'un graphe orienté
- La structure choisie est un graphe orienté
  - Un graphe orienté est un ensemble de noeuds et des arêtes entre les noeuds
- L'algorithme choisi est la fermeture transitive
  - La fermeture transitive construit un autre graphe tel que chaque arête correspond avec un chemin dans le graphe original
- Cet exemple est expliqué en détail dans la version anglaise du livre
  - Voir la section 6.8.1

© 2007 P. Van Roy. All rights reserved.

Fermeture transitive d’un graphe





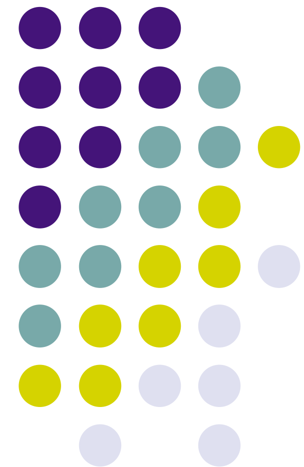
Les noeuds: {1,2,3,4,5,6}

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

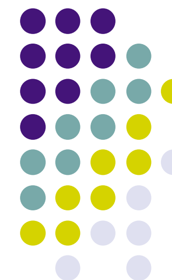
© 2007 P. Van Roy. All rights reserved.

Résumé



© 2007 P. Van Roy. All rights reserved.

Résumé



- Nous avons donné plusieurs exemples de types abstraits: des collections indexées et des matrices
- Collections indexées
  - Tuple
  - Enregistrement
  - Tableau (avec état, indexes sont des entiers)
  - Dictionnaire (avec état, indexes sont des litéraux)
- Matrices
  - Addition et multiplication
  - Avec plusieurs représentations
  - Comparaison des algorithmes

© 2007 P. Van Roy. All rights reserved.