

FSAB1402: Informatique 2

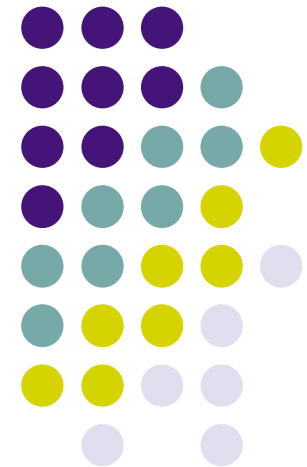
La Concurrence Déclarative



Département d'Ingénierie Informatique, UCL

Peter Van Roy

pvr@info.ucl.ac.be



Ce qu'on va voir aujourd'hui



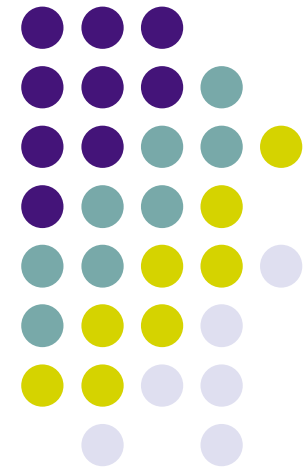
- La concurrence déclarative
 - On peut prendre un programme déclaratif et le rendre concurrent simplement en ajoutant des fils, sans changer autre chose
- La programmation multi-agent
- Quelques regards sur le cours
 - Réflexions sur les paradigmes de programmation
 - Un exercice de sémantique
 - La complexité
- Des consignes et des conseils pour l'examen

Suggestions de lecture pour ce cours



- Chapitre 4 (section 4.2)
 - Programmer avec les fils
- Chapitre 4 (section 4.3)
 - Agents, flots, pipelines
- Chapitre 2 (section 2.6)
 - Du langage noyau au langage pratique

La concurrence déclarative





La concurrence déclarative

- Les programmes multi-agents que nous avons vus hier sont **déterministes**
 - Avec les mêmes entrées, ils donnent les mêmes résultats
 - L'agent Trans, avec l'entrée 1|2|3|_, donne toujours la sortie 1|4|9|_
- Dans ces programmes, la concurrence ne change pas la valeur du résultat, mais uniquement l'ordre du calcul (**quand** le résultat est calculé)
 - Cette propriété facilite de beaucoup la programmation
 - On peut ajouter des fils à un programme existant sans que le résultat change (la "**transparence**" de la concurrence)
- Cette propriété est vraie uniquement pour la programmation déclarative
 - Elle n'est pas vraie pour la programmation avec état

La concurrence déclarative est transparente (1)



```
fun {Map Xs F}
  case Xs
  of nil then nil
  [] X|Xr then
    {F X} | {Map Xr F}
  end
end
```

La concurrence déclarative est transparente (2)



```
fun {CMap Xs F}
  case Xs
  of nil then nil
  [] X|Xr then
    thread {F X} end | {CMap Xr F}
  end
end
```

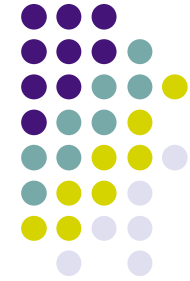
La concurrence déclarative est transparente (3)



```
fun {CMap Xs F}
  case Xs
  of nil then nil
  [] X|Xr then
    thread {F X} end | {CMap Xr F}
  end
end
```

thread ... end
peut être utilisé
comme expression

La concurrence déclarative est transparente (4)



```
fun {CMap Xs F}
  case Xs
  of nil then nil
  [] X|Xr then
    thread {F X} end | {CMap Xr F}
  end
end
```

- Qu'est-ce qui se passe si on fait:
declare F
{Browse {CMap [1 2 3 4] F}}

La concurrence déclarative est transparente (5)



```
fun {CMap Xs F}
  case Xs
  of nil then nil
  [] X|Xr then
    thread {F X} end | {CMap Xr F}
  end
end
```

- Le browser montre [_ _ _ _]
 - CMap calcule le “squelette” de la liste
 - Les nouveaux fils attendent que F soit lié

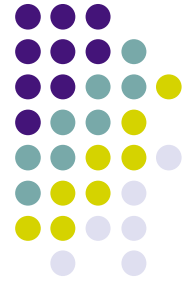
La concurrence déclarative est transparente (6)



```
fun {CMap Xs F}
  case Xs
  of nil then nil
  [] X|Xr then
    thread {F X} end | {CMap Xr F}
  end
end
```

- Qu'est-ce qui se passe si on ajoute:
F = **fun** {\$ X} X+1 **end**

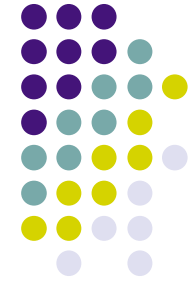
La concurrence déclarative est transparente (7)



```
fun {CMap Xs F}
  case Xs
  of nil then nil
  [] X|Xr then
    thread {F X} end | {CMap Xr F}
  end
end
```

- Le browser montre [2 3 4 5]

La concurrence pour les nuls (1)



- On peut ajouter des fils à un programme déclaratif existant sans que le résultat change
- Par conséquence, il est très facile de prendre un programme déclaratif et de le rendre concurrent
- Il suffit d'insérer l'instruction **thread** ... **end** là où on a besoin de la concurrence
- **Attention**: la concurrence pour les nuls ne marche qu'avec un programme déclaratif (sans cellules)!
 - Cela ne marchera pas pour Java, par exemple

La concurrence pour les nuls (2)



```
fun {Fib X}
  if X==0 then 0
  elseif X==1 then 1
  else
    thread {Fib X-1} end + {Fib X-2}
  end
end
```

Pourquoi cela marche?

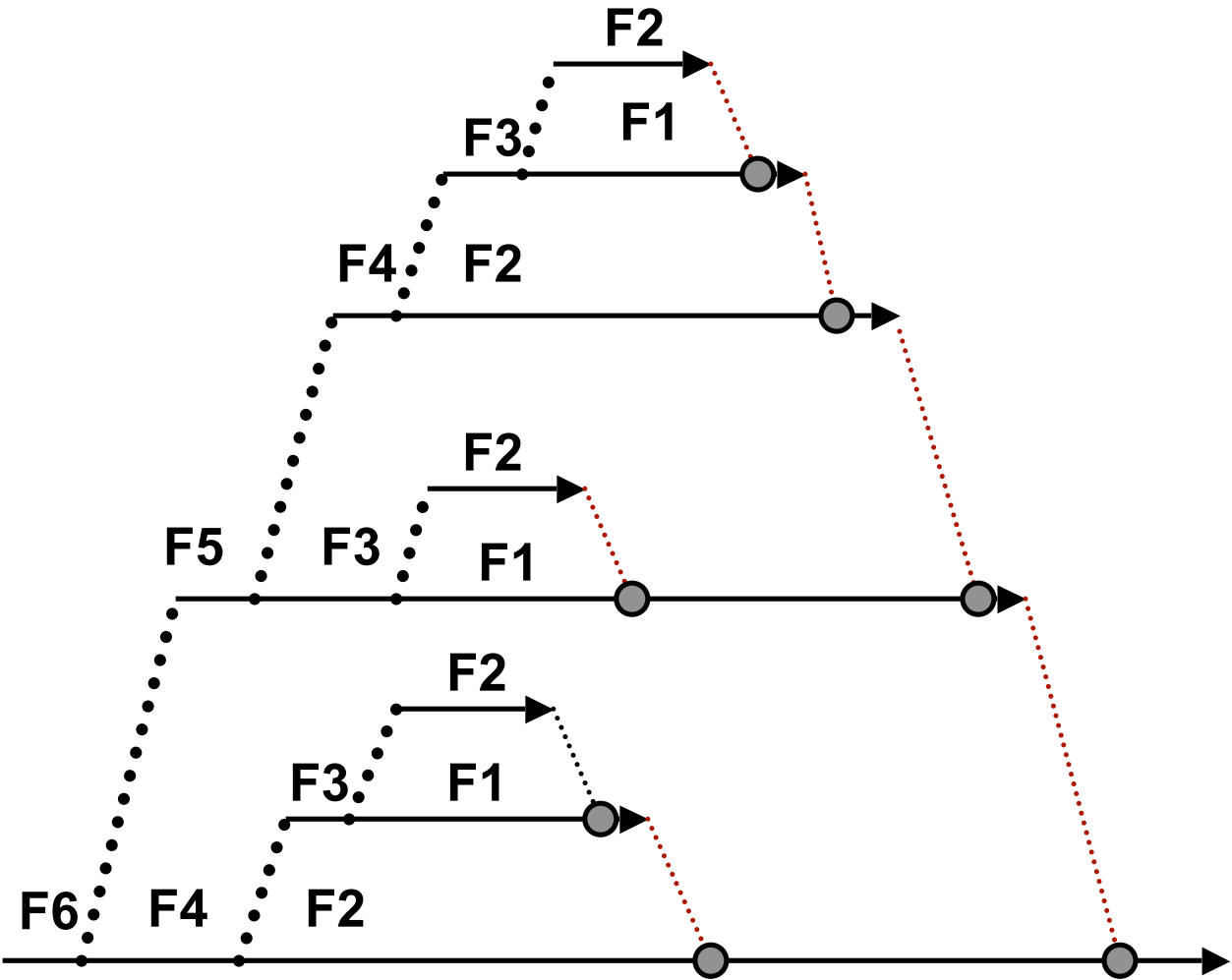


```
fun {Fib X}
  if X==0 then 0 elseif X==1 then 1
  else F1 F2 in
    F1 = thread {Fib X-1} end
    F2 = {Fib X-2}
    F1 + F2
  end
end
```

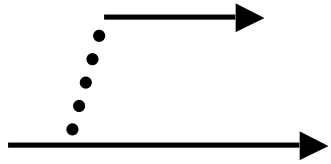
Dépendance dataflow

Pour comprendre pourquoi cela marche, voici le programme en partie en langage noyau

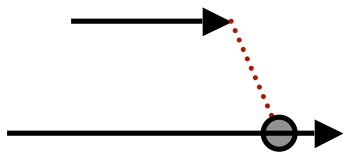
Exécution de {Fib 6}



Créer un thread



Synchroniser avec le résultat

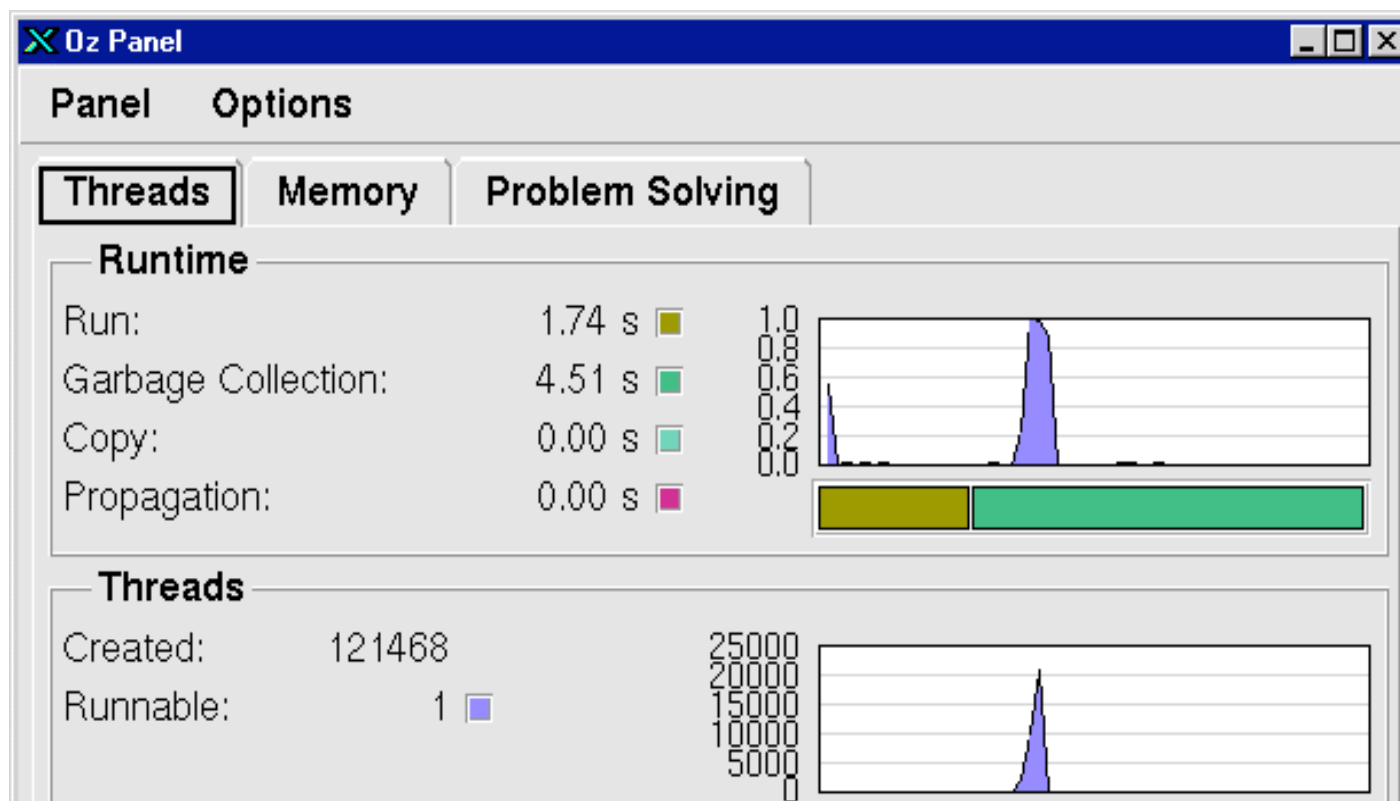


Thread actif

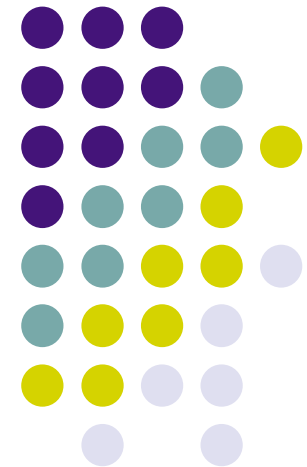




Observer l'exécution de Fib



La programmation multi-agent



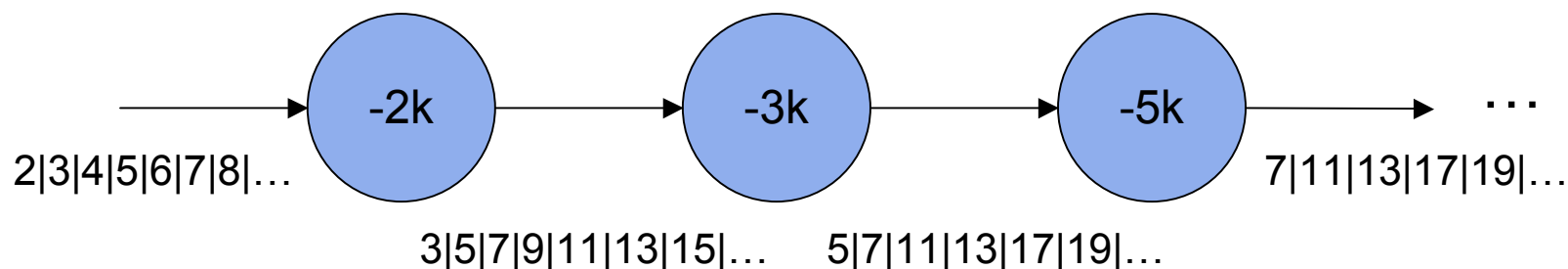


Encore des agents!

- Hier nous avons vu quelques exemples simples de programmes multi-agents
 - Producteur-consommateur
 - Producteur-transformateur-consommateur (pipeline)
- Regardons maintenant un exemple plus sophistiqué

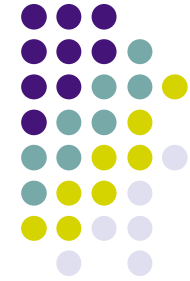


Le crible d'Eratosthènes



- Le crible d'Eratosthènes est un algorithme pour faire la séquence des nombres premiers
- On commence avec une séquence d'entiers, on la passe par un pipeline d'agents dont chaque agent enlève les multiples du premier élément

Un agent pour enlever des multiples



- Voici un programme qui enlève les multiples de k:

```
fun {Filtre Xs K}  
  case Xs of X|Xr then  
    if X mod K  $\neq$  0 then X|{Filtre Xr K}  
    else {Filtre Xr K} end  
  else nil  
  end  
end
```

- Pour en faire un agent, il faut le mettre dans un fil:

```
thread Ys={Filtre Xs K} end
```



Le programme principal

- Voici le programme principal:

```
fun {Crible Xs}  
  case Xs  
  of nil then nil  
  [] X|Xr then X|{Crible thread {Filtre Xr X} end}  
  end  
end
```

```
declare Xs Ys in  
thread Xs={Prod 2} end  
thread Ys={Crible Xs} end  
{Browse Ys}
```



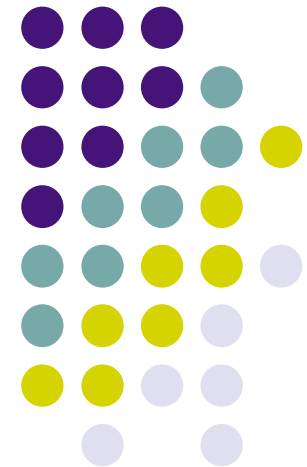
Une optimisation

- Sinon on crée beaucoup trop d'agents!

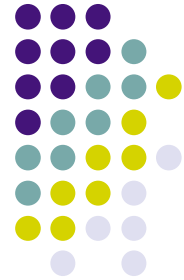
```
fun {Crible2 Xs M}  
  case Xs  
  of nil then nil  
  [] X|Xr then  
    if X=<M then  
      X|{Crible2 thread {Filtre Xr X} end M}  
    else Xs end  
  end  
end
```

- On appelle alors {Crible2 Xs 316} pour une liste avec des nombres premiers jusqu'au 100000 (pourquoi?)

Réflexions sur les paradigmes de programmation



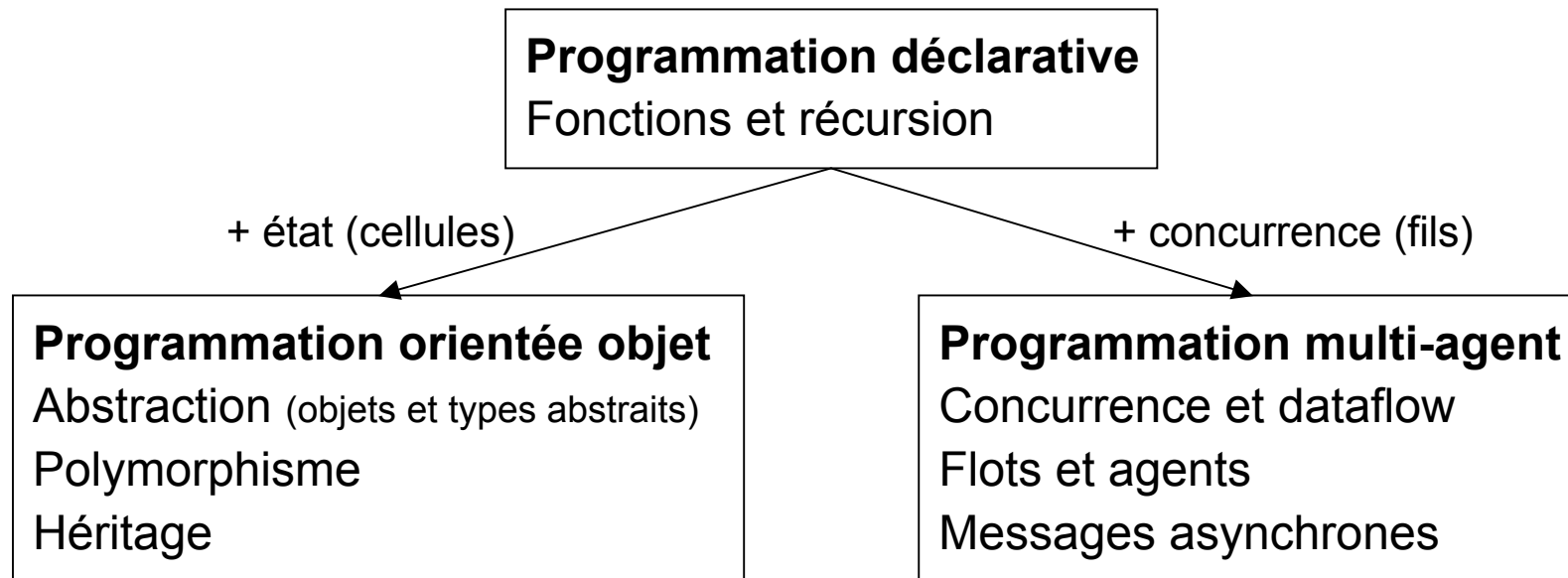
Les paradigmes de FSAB1402



- Dans ce cours nous avons vu quelques uns des concepts les plus importants de la programmation
- Nous avons aussi vu quelques paradigmes de programmation
 - Programmation déclarative (programmation fonctionnelle)
 - Programmation avec état
 - Programmation orientée objet
 - Programmation concurrente avec dataflow
 - Programmation multi-agent
- Il y a beaucoup d'autres paradigmes intéressants!
 - Programmation concurrente par envoi de messages
 - Programmation concurrente par état partagé
 - Programmation par composants logiciels
 - Programmation logique
 - Programmation par contraintes
 - ...



Les trois “mondes” du cours



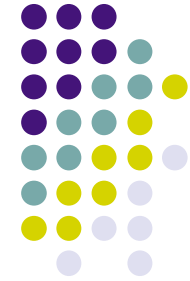
- Dans ce cours, nous avons vu trois “mondes” très différents, chacun avec sa manière de penser
 - Pour voir comment l’orienté objet et le multi-agent peuvent être combiné, il faut suivre le cours INGI1131!

Le langage noyau du modèle déclaratif



- $\langle s \rangle ::=$ **skip**
 - | $\langle s \rangle_1 \langle s \rangle_2$
 - | **local** $\langle x \rangle$ **in** $\langle s \rangle$ **end**
 - | $\langle x \rangle_1 = \langle x \rangle_2$
 - | $\langle x \rangle = \langle v \rangle$
 - | **if** $\langle x \rangle$ **then** $\langle s \rangle_1$ **else** $\langle s \rangle_2$ **end**
 - | $\{ \langle x \rangle \langle y \rangle_1 \dots \langle y \rangle_n \}$
 - | **case** $\langle x \rangle$ **of** $\langle p \rangle$ **then** $\langle s \rangle_1$ **else** $\langle s \rangle_2$ **end**
- $\langle v \rangle ::=$ $\langle \text{number} \rangle$ | $\langle \text{procedure} \rangle$ | $\langle \text{record} \rangle$
- $\langle \text{number} \rangle ::=$ $\langle \text{int} \rangle$ | $\langle \text{float} \rangle$
- $\langle \text{procedure} \rangle ::=$ **proc** $\{ \$ \langle x \rangle_1 \dots \langle x \rangle_n \}$ $\langle s \rangle$ **end**
- $\langle \text{record} \rangle, \langle p \rangle ::=$ $\langle \text{lit} \rangle (\langle f \rangle_1 : \langle x \rangle_1 \dots \langle f \rangle_n : \langle x \rangle_n)$

Le langage noyau du modèle orienté objet



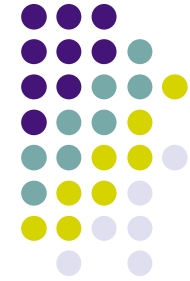
- $\langle s \rangle ::=$
 - skip**
 - $\langle s \rangle_1 \langle s \rangle_2$
 - local** $\langle x \rangle$ **in** $\langle s \rangle$ **end**
 - $\langle x \rangle_1 = \langle x \rangle_2$
 - $\langle x \rangle = \langle v \rangle$
 - if** $\langle x \rangle$ **then** $\langle s \rangle_1$ **else** $\langle s \rangle_2$ **end**
 - $\{ \langle x \rangle \langle y \rangle_1 \dots \langle y \rangle_n \}$
 - case** $\langle x \rangle$ **of** $\langle p \rangle$ **then** $\langle s \rangle_1$ **else** $\langle s \rangle_2$ **end**
 - $\{ \text{NewCell } \langle x \rangle \langle y \rangle \}$
 - $\langle y \rangle := \langle x \rangle$
 - $\langle x \rangle = @ \langle y \rangle$
 - try** $\langle s \rangle_1$ **catch** $\langle x \rangle$ **then** $\langle s \rangle_2$ **end**
 - raise** $\langle x \rangle$ **end**

Modèle
déclaratif

Extension avec l'état
et les exceptions

- $\langle v \rangle ::= \langle \text{number} \rangle \mid \langle \text{procedure} \rangle \mid \langle \text{record} \rangle$
- $\langle \text{number} \rangle ::= \langle \text{int} \rangle \mid \langle \text{float} \rangle$
- $\langle \text{procedure} \rangle ::= \text{proc } \{ \$ \langle x \rangle_1 \dots \langle x \rangle_n \} \langle s \rangle \text{ end}$
- $\langle \text{record} \rangle, \langle p \rangle ::= \langle \text{lit} \rangle (\langle f \rangle_1 : \langle x \rangle_1 \dots \langle f \rangle_n : \langle x \rangle_n)$

Le langage noyau du modèle multi-agent



- $\langle s \rangle ::=$
 - skip**
 - $\langle s \rangle_1 \langle s \rangle_2$
 - local** $\langle x \rangle$ **in** $\langle s \rangle$ **end**
 - $\langle x \rangle_1 = \langle x \rangle_2$
 - $\langle x \rangle = \langle v \rangle$
 - if** $\langle x \rangle$ **then** $\langle s \rangle_1$ **else** $\langle s \rangle_2$ **end**
 - $\{ \langle x \rangle \langle y \rangle_1 \dots \langle y \rangle_n \}$
 - case** $\langle x \rangle$ **of** $\langle p \rangle$ **then** $\langle s \rangle_1$ **else** $\langle s \rangle_2$ **end**
 - thread** $\langle s \rangle$ **end**
- $\langle v \rangle ::= \langle \text{number} \rangle \mid \langle \text{procedure} \rangle \mid \langle \text{record} \rangle$
- $\langle \text{number} \rangle ::= \langle \text{int} \rangle \mid \langle \text{float} \rangle$
- $\langle \text{procedure} \rangle ::= \text{proc } \{ \$ \langle x \rangle_1 \dots \langle x \rangle_n \} \langle s \rangle \text{end}$
- $\langle \text{record} \rangle, \langle p \rangle ::= \langle \text{lit} \rangle (\langle f \rangle_1 : \langle x \rangle_1 \dots \langle f \rangle_n : \langle x \rangle_n)$

Modèle
déclaratif

Extension avec
les fils



D'autres extensions?

- On peut ajouter tout: l'état, les exceptions et les fils
 - Cela donne le **modèle concurrent par état partagé**
 - C'est assez compliqué à programmer
- Une autre possibilité est d'ajouter des canaux de communication à la place de l'état
 - Cela donne le **modèle concurrent par envoi de messages**
 - C'est un très bon modèle!
- Il y a encore d'autres possibilités
 - Etendre le modèle déclaratif avec **le calcul "paresseux"**
- Voir le cours INGI1131!

Paradigmes de programmation



Programmation déclarative

Programmation fonctionnelle stricte, Scheme, ML

Programmation logique déterministe

+ concurrence

+ synchronisation selon besoin

Concurrence dataflow (déclarative)

Prog. fonctionnelle paresseuse, Haskell

+ choix nondéterministe

Programmation logique concurrente

+ traitement d'exceptions

+ état explicite

Programmation orientée objet (OO), Java, C++, Smalltalk

+ recherche

Prog. logique classique, Prolog

- Ce schéma donne un résumé des différents paradigmes avec les relations entre eux
- Chaque paradigme a ses avantages et désavantages et un domaine où il est le meilleur

Programmation OO concurrente (envoi de messages, Erlang, E) (état partagé, Java)

+ espaces de calcul

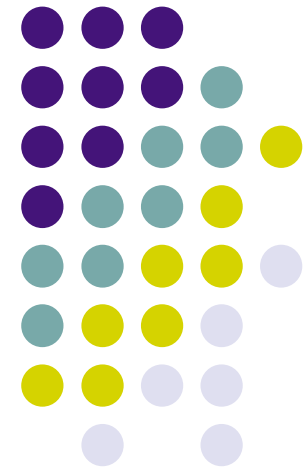
Programmation par contraintes

La coexistence des paradigmes



- Chaque paradigme a sa place
 - Avec plus de concepts on peut exprimer plus, mais le raisonnement devient plus compliqué
 - Avec moins de concepts on peut satisfaire des conditions d'utilisation plus stricte
- Les différents paradigmes ne sont pas meilleurs ou pires, mais simplement différents
 - Dans vos programmes, je vous conseille de bien réfléchir et de choisir le paradigme approprié
- Maintenant, je vous conseille de **relire le début du premier cours!**
 - Pourquoi le cours est organisé autour des concepts

Un exercice de sémantique





Facilité avec la sémantique

- La sémantique est une partie importante de ce cours
 - Il ne s'agit pas seulement de savoir programmer avec les concepts
 - Il s'agit de les comprendre, et donc de connaître leur sémantique
- J'attends que vous puissiez faire des calculs rapides avec la sémantique des programmes
- Il faut faire des exercices (écrire les pas à la main) pour gagner cette facilité



L'énoncé

- Quel est l'état à la fin de l'exécution de:

```
local MakeBumper B Y in  
  fun {MakeBumper}  
    C={NewCell 0}  
  in  
    fun {$} C:=@C+1 @C end  
  end  
  B={MakeBumper}  
  Y={B}  
end
```

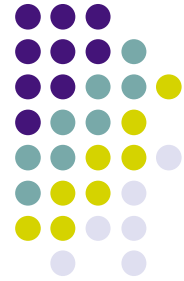
- Montrez quelques pas d'exécution représentatifs



Vers le langage noyau...

```
S1 {  
  local MakeBumper B Y in  
    S2 {  
      proc {MakeBumper R}  
        C={NewCell 0}  
      in  
        R=proc {$ K} C:=@C+1 K=@C end  
      end  
      {MakeBumper B}  
      {B Y}  
    }  
  end
```

→ **Solution au tableau**



L'état final

Une cellule



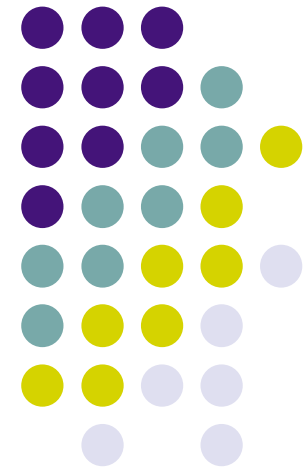
([], { $m=p_1$, $b=p_2$, $y=1$, $c=\xi$, $i=0$, **$c:y$** })

avec:

$p_1 = (\text{proc } \{ \$ R \} \text{ local } C = \{ \text{NewCell } 0 \} \text{ in } \dots \text{ end end, } \{ \})$

$p_2 = (\text{proc } \{ \$ K \}$
 local X Y **in** X=@C Y=X+1 C:=Y K=@C **end**
 end, {C → c})

La complexité





Conseils

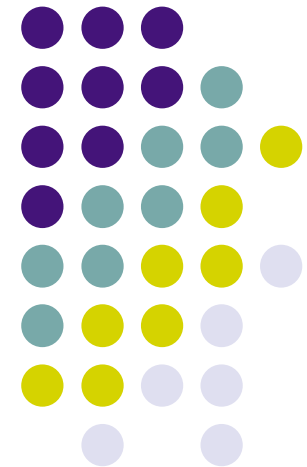
- Ne pas utiliser des équations de récurrence
 - Nous n'avons pas fait des exercices pour cela
- Utiliser un raisonnement sur le nombre d'appels et la taille des structures de données
- Attention: il y a trois concepts orthogonaux (voir transparent suivant)



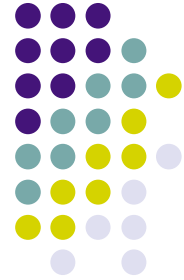
Les trois axes

- Le temps ou l'espace
 - Complexité temporelle (temps utilisé)
 - Complexité spatiale (taille mémoire utilisée)
- Les bornes:
 - Supérieure (notation O)
 - Inférieure (notation Ω)
 - λ Supérieure et inférieure (notation Θ)
- La distribution des entrées (**toujours** en fonction de la taille $n!$)
 - Pire cas: ces entrées qui donnent la pire borne
 - Meilleur cas: idem mais la meilleure borne
 - Moyenne cas: cas "typique" des entrées
 - Question: Pour une fonction de liste, pourquoi prendre le cas d'une liste vide est faux pour avoir une complexité en meilleur cas?

Consignes et conseils pour l'examen



L'examen

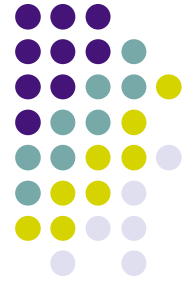


- L'examen sera de 3h, à livre fermé
- Il y aura une division égale entre théorie et pratique
 - Attention à la **précision** pour la théorie (voir le livre pour les définitions!)
 - Attention à la **syntaxe** pour la pratique!
- Il y aura certainement une question sur la **sémantique**
 - Un exercice où vous devez faire l'exécution d'un programme
 - Attention à ne pas sombrer dans les détails!
 - Sur le site Web il y a **une ancienne question d'examen avec solution** (faite par Damien Saucez et Anh Tuan Tang Mac)
- La matière: les séances magistrales et les séances pratiques
 - Tout le contenu du livre (les parties non-vues dans les séances seront considérées comme des **bonus**)
 - Des définitions précises et des exemples supplémentaires se trouvent dans le livre
- Les notes ne seront pas sur une courbe
 - J'espère pouvoir vous donner tous de bonnes notes



La sémantique

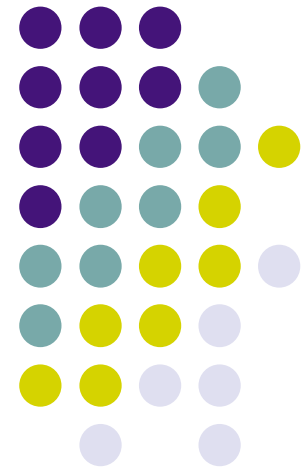
- Le langage noyau
- L 'environnement contextuel
- La définition et l'appel des procédures
 - Attention à l'environnement pendant l'exécution de la procédure: ce n'est pas le même que l'environnement à l'appel!
- La définition et l'exécution des cellules
 - Une cellule est une paire, qui vit dans la mémoire à affectation multiple
- Exécuter des petits programmes avec la sémantique
 - Pas montrer tous les détails, mais montrer les choses importantes



Le formulaire pour la syntaxe

- Vous pouvez venir à l'examen avec un formulaire pour la syntaxe
 - Le but est que vous ne faites pas d'erreurs de syntaxe dans vos réponses aux questions
 - Le formulaire doit être **écrit à la main** et **rendu avec l'examen**, mais il ne sera pas coté
- **Une page**, contenant uniquement des fragments de programme et des règles de grammaire
 - Fragments en Oz et en Java
 - Aucun mot écrit en langue naturelle
 - Aucune équation mathématique

Résumé



Résumé



- La concurrence déclarative
 - On peut prendre n'importe quel programme déclaratif et le rendre concurrent (“concurrence pour les nuls”)
 - Attention: ceci ne marche pas pour les programmes avec des cellules (l'état)!
- La programmation multi-agent
 - Un exemple plus compliqué: le crible d'Eratosthènes
- Un exercice de sémantique
- Les paradigmes de programmation
 - Une réflexion sur le contenu du cours
- Des consignes et des conseils pour l'examen
 - Attention aux définitions précises!
 - Il y aura certainement un exercice sur la sémantique