



# Contrôle – MR2A Informatique

## Module MMPFPSD

mercredi 18 janvier 2006

A. BEUGNARD, F. DAGNAT

### Correction

## Consignes

**Tout document est autorisé !**

Cet examen est prévu pour durer deux heures. Son but est de vérifier votre compréhension, veillez donc à ce que vos explications soient claires et concises.

**Merci d'avance et bon travail !**

Exercice 1 ( <i>Liaison dynamique</i> )	[25 mn – 4,1/20]	1
Exercice 2 ( <i>La notion de référence</i> )	[44 mn – 7,3/20]	2
Exercice 3 ( <i>Un petit méta-modèle objet</i> )	[36 mn – 5,9/20]	6
Exercice 4 ( <i>Un <math>\pi</math>-calcul un peu spécial ...</i> )	[17 mn – 2,8/20]	8

## Exercice 1 (*Liaison dynamique*) **[25 mn – 4,1/20]**

Les expériences présentées en cours sur les différentes interprétations de la redéfinition et de la surcharge se limitent à la *liaison dynamique simple*, c'est-à-dire sont fonction du type effectif de *l'objet récepteur*. Mettez en place une expérience qui met en évidence les différences entre liaison dynamique simple et liaison dynamique multiple. La liaison dynamique multiple utilise le type effectif des objets passés *en paramètre* en non leurs types déclarés.

### Question 1 **[20 mn – 3,3/20]**

▷ **Décrire l'expérience.**

Le point clé était de faire apparaître une différence entre les types déclarés et effectifs des paramètres. Dans une expérience minimaliste, un seul type de récepteur suffisait, avec 2 types de paramètres.

Pour reprendre les classes vues en cours. Up, Top, Middle. Des variables u (Up), t (Top), m (Middle) et tm (Middle déclaré Top). La méthode m est déclarée avec les signatures m(Top) et m(Middle) On réalise les appels u.m(t), u.m(m). et u.m(tm).

### Question 2

[5 mn – 0,8/20]

▷ Montrer le résultat attendu par votre expérience dans le cas d'un langage à liaison dynamique simple et dans le cas d'un langage à liaison dynamique multiple.

appels	t	m	tm
u.m(x) simple dispatch	m(Top)	m(Middle)	m(Top)
u.m(x) dispatch multiple	m(Top)	m(Middle)	m(Middle)

C'est la troisième colonne du résultat qui met en évidence les différences...

### Exercice 2 (*La notion de référence*)

[44 mn – 7,3/20]

Dans la majorité des langages, il est nécessaire d'introduire, en plus, de la notion de variable, la notion de référence. Une référence va modéliser une *cellule mémoire*. Ainsi, on pourra étudier les manipulations de variables qui peuvent être modifiées (par l'opération d'affectation).

Pour cela, on se place dans le cadre du  $\lambda$ -calcul que l'on étend par trois *opérations prédéfinies* :

1. **unit** : une constante qui représente le « rien »,
2. **ref** : qui prend en paramètre une valeur  $v$  et renvoie l'adresse d'une nouvelle *cellule mémoire* qui contiendra  $v$ ,
3. **!** : qui prend en paramètre une *adresse*  $a$  et renvoie la valeur contenue dans la *cellule mémoire* d'adresse  $a$ ,
4. **:=<sup>1</sup>** : qui prend en paramètre une *adresse*  $a$  et une valeur  $v$ ; il modifie la *cellule mémoire* d'adresse  $a$  qui contiendra maintenant  $v$  et retourne **unit**.

Ce  $\lambda$ -calcul étendu que l'on notera  $\lambda_{ref}$ -calcul suit donc la syntaxe suivante :

$M ::=$	$x$	Une variable
	$M M$	Une application
	$\lambda x.M$	Une fonction
	<b>unit</b>	La constante <b>unit</b>
	<b>ref</b> $M$	Une création de référence
	<b>!</b> $M$	Accès à la valeur d'une référence
	$M := M$	Affectation
	$a$	Une adresse

L'ensemble des variables est noté  $\mathcal{Var}$ , celui des adresses  $\mathcal{A}$  et celui des termes  $\mathcal{T}$ .

<sup>1</sup>Cet opérateur est supposé infixé, c'est-à-dire qu'il s'utilise sous la forme  $a := v$

**Remarque :**

Un terme de  $\lambda_{ref}$  écrit par un « programmeur » ne contiendra pas d'adresse. Les adresses n'apparaîtront que par des réductions. En effet, la définition informelle des références indique que l'opération  $\text{ref}M$  renvoie une adresse.

Une valeur du  $\lambda_{ref}$ -calcul est un terme qui ne se réduit plus défini par la grammaire  $v ::= \lambda x.M \mid \text{unit} \mid a$ . Nous noterons  $\mathcal{V}$ , l'ensemble des valeurs closes (celles qui ne contiennent pas de variable libre).

Pour exprimer la sémantique de  $\lambda_{ref}$ , il faut modifier légèrement les règles de réduction. En effet, il faut introduire la *mémoire* qui contiendra les cellules. Une mémoire ne pourra mémoriser que des valeurs closes et est donc une fonction de l'ensemble des adresses vers celui des variables closes (soit  $\mu : \mathcal{A} \rightarrow \mathcal{V}$ ). Une mémoire  $\mu$  sera notée  $\{a_1 \mapsto v_1, \dots, a_n \mapsto v_n\}$ , son domaine  $\text{dom}(\mu)$  et l'ajout d'une liaison  $\mu :: \{a \mapsto v\}$ . La valeur associée à une cellule est donnée par la dernière liaison ajoutée :

$$\mu :: \{a \mapsto v\}(b) = \begin{cases} v & \text{si } b = a \\ \mu(b) & \text{sinon} \end{cases}$$

Enfin, l'opérateur de réduction  $\rightarrow$  est étendu pour porter sur des paires  $(M, \mu)$ . La réduction  $(\rightarrow_C (\mathcal{T} \times \mathcal{M}) \times (\mathcal{T} \times \mathcal{M}))$ . Ainsi :

$$M_1 \mid \mu_1 \rightarrow M_2 \mid \mu_2$$

signifie que la réduction du terme  $M_1$  avec une mémoire  $\mu_1$  mène au terme  $M_2$  avec une mémoire modifiée  $\mu_2$ .

**Question 1****[2 mn – 0,3/20]**

▷ Donner la règle de  $\beta$ -réduction étendue (avec une mémoire).

$$\text{App} : \lambda x.M_1 M_2 \mid \mu \rightarrow$$

$$\text{APP} : \lambda x.M_1 M_2 \mid \mu \rightarrow [x \mapsto M_2]M_1 \mid \mu$$

La question est plutôt bien abordée. La principale erreur est d'ajouter la variable  $x$  en mémoire. Attention, mes variables ne sont pas stockées en mémoire ! Seules les références sont stockées en mémoire.

**Question 2****[20 mn – 3,3/20]**

L'ajout de constantes (comme dans le cas de  $\lambda_{ref}$ ) dans le  $\lambda$ -calcul ne modifie pas sa sémantique. Il suffit de définir le comportement des nouvelles constantes par des règles de réduction appelées  $\delta$ -réductions.

▷ Compléter les règles de  $\delta$ -réductions des opérations `ref`, `!` et `:=` ci-dessous.

$$\text{CreateCell} : \frac{\dots}{\text{ref } v \mid \mu \rightarrow \dots}$$

$$\text{Ref} : \frac{\dots}{\text{ref } M \mid \mu \rightarrow \dots}$$

$$\text{ReadCell} : \frac{\dots}{!a \mid \mu \rightarrow \dots}$$

$$\text{Deref} : \frac{\dots}{!M \mid \mu \rightarrow \dots}$$

$$\text{WriteCell} : \frac{\dots}{a := v \mid \mu \rightarrow \dots}$$

$$\text{AssignL} : \frac{\dots}{M_1 := M_2 \mid \mu \rightarrow \dots}$$

$$\text{AssignR} : \frac{\dots}{M_1 := M_2 \mid \mu \rightarrow \dots}$$

$$\text{CREATECELL} : \frac{a \notin \text{dom}(\mu)}{\text{ref } v \mid \mu \rightarrow a \mid \mu :: \{a \mapsto v\}}$$

$$\text{REF} : \frac{M \mid \mu \rightarrow M' \mid \mu'}{\text{ref } M \mid \mu \rightarrow \text{ref } M' \mid \mu'}$$

$$\text{READCELL} : \frac{a \in \text{dom}(\mu)}{!a \mid \mu \rightarrow \mu(a) \mid \mu}$$

$$\text{DEREF} : \frac{M \mid \mu \rightarrow M' \mid \mu'}{!M \mid \mu \rightarrow !M' \mid \mu'}$$

$$\text{WRITECELL} : \frac{a \in \text{dom}(\mu)}{a := v \mid \mu \rightarrow \text{unit} \mid \mu :: \{a \mapsto v\}}$$

$$\text{ASSIGNL} : \frac{M_1 \mid \mu \rightarrow M'_1 \mid \mu'}{M_1 := M_2 \mid \mu \rightarrow M'_1 := M_2 \mid \mu'}$$

$$\text{ASSIGNR} : \frac{M_2 \mid \mu \rightarrow M'_2 \mid \mu'}{M_1 := M_2 \mid \mu \rightarrow M_1 := M'_2 \mid \mu'}$$

Le barème était de (0,7) (resp. (0,3)) par règle pour les règles CREATECELL, READCELL et WRITECELL (resp. REF, Deref, ASSIGNL et ASSIGNR).

Globalement, les règles CREATECELL, READCELL et WRITECELL ont été comprises et exprimées à peu près correctement. Par contre, les quatre autres n'ont pas été bien abordées :

- Ces règles doivent être complètes (*i.e.* couvrir tous les cas possibles (exception faite des termes dont on souhaite qu'ils ne se réduisent pas – en général, ce sont des erreurs voir question suivante). Par exemple, d'après la grammaire, les références sont de la forme `ref M`. Alors, il y a deux cas :

1.  $M$  est une valeur (règle CREATECELL) qu'il faut alors stocker en mémoire en allouant une nouvelle adresse (condition  $a \notin \text{dom}(\mu)$ );
2.  $M$  peut se réduire en  $M'$  (règle REF) cette réduction peut avoir un effet sur la mémoire qu'il faut alors répercuter.

- Notre sémantique opérationnelle est dite à *petits pas* puisqu'il décrit toutes les étapes de réduction. Certains ont exigé que, par exemple dans REF,  $M$  se réduisent en une valeur (parfois en une étape, parfois en plusieurs). Cela pose deux problèmes :

1. En plusieurs étapes ( $M \rightarrow^* v$ ) : on ne peut pas mélanger des règles de petits pas (CREATECELL, READCELL et WRITECELL) avec des règles de plus grand pas.
2. En une étape ( $M \rightarrow v$ ) : et si  $M$  se réduit en autre chose qu'une valeur ?

**Question 3****[6 mn – 1/20]**

Un certain nombre de termes, qui ne sont pas des valeurs, ne se réduisent pas. Ils mènent à des réductions non souhaitées que nous allons éliminer en construisant un système de type pour  $\lambda_{ref}$ . Le calcul devient typé, c'est-à-dire que l'introduction de variable doit déclarer son type,  $\lambda x.M$  devient  $\lambda x : t.M$

**Remarque :**

*Dans un cadre typé, nous n'autorisons pas les variables libres globales. C'est-à-dire le terme en cours de réduction n'a pas de variable libre.*

▷ **Quelles sont, à votre avis, les erreurs *interdites* (i.e. les termes irréductibles que nous souhaitons éliminer) ?**

Les erreurs *interdites* possibles sont :

1. Application d'une valeur qui n'est pas une fonction :

$$\text{APPERR1} : \text{unit } M \mid \mu \rightarrow \mathbf{error} \qquad \text{APPERR2} : a \ M \mid \mu \rightarrow \mathbf{error}$$

2. Utilisation d'une valeur qui n'est pas une adresse pour lire une référence :

$$\text{DEREFERR1} : !\text{unit} \mid \mu \rightarrow \mathbf{error} \qquad \text{DEREFERR2} : !\lambda x.M \mid \mu \rightarrow \mathbf{error}$$

3. Utilisation d'une valeur qui n'est pas une adresse pour une affectation :

$$\text{ASSIGNLERR1} : \text{unit} := M \mid \mu \rightarrow \mathbf{error} \qquad \text{ASSIGNLERR2} : \lambda x.M_1 := M_2 \mid \mu \rightarrow \mathbf{error}$$

Le barème était de (0,2) pour les erreurs d'application et de (0,4) pour chacune des deux autres.

La question a été plutôt mal abordée, vous vous contentez, en général, d'une description informelle ce qui n'est pas acceptable.

Les variables libres évoquées par la remarque sont déjà éliminées par le cadre usuel du  $\lambda$ -calcul typé. J'attendais plutôt les nouveaux termes qui posent des problèmes.

**Question 4****[16 mn – 2,7/20]**

Pour éliminer ces erreurs, nous allons utiliser les types définis par  $t ::= \text{Unit} \mid t \rightarrow t \mid \text{Ref } t$  et des règles de typage de la forme  $\Gamma, \Sigma \vdash M : t$  où  $\Gamma$  est l'environnement de typages des variables et  $\Sigma$  celui des adresses.

Les règles usuelles sont inchangées (si ce n'est l'ajout du  $\Sigma$ ) et la constante `unit` est typée par le type `Unit` :

$$\text{T}_{\text{VAR}} : \frac{x \in \text{dom}(\Gamma)}{\Gamma, \Sigma \vdash x : \Gamma(x)}$$

$$\text{T}_{\text{FUN}} : \frac{\Gamma :: \{x \mapsto t\}, \Sigma \vdash M : t'}{\Gamma, \Sigma \vdash \lambda x : t.M : t \rightarrow t'}$$

$$\text{T}_{\text{APP}} : \frac{\Gamma, \Sigma \vdash M_1 : t \rightarrow t' \quad \Gamma, \Sigma \vdash M_2 : t}{\Gamma, \Sigma \vdash M_1 \ M_2 : t'}$$

$$\text{T}_{\text{UNIT}} : \Gamma, \Sigma \vdash \text{unit} : \text{Unit}$$

▷ Compléter les règles suivantes qui complèterons le système de type.

$$\begin{array}{l}
 \mathbf{TLoc} : \frac{\dots}{\Gamma, \Sigma \vdash a : \dots} \qquad \mathbf{TRef} : \frac{\dots}{\Gamma, \Sigma \vdash \text{ref } M : \dots} \qquad \mathbf{TDeref} : \frac{\dots}{\Gamma, \Sigma \vdash !M : \dots} \\
 \\
 \mathbf{TAssign} : \frac{\dots}{\Gamma, \Sigma \vdash M_1 := M_2 : \dots}
 \end{array}$$

$$\begin{array}{l}
 \mathbf{TLoc} : \frac{a \in \text{dom}(\Sigma)}{\Gamma, \Sigma \vdash a : \text{Ref } \Sigma(a)} \qquad \mathbf{TREF} : \frac{\Gamma, \Sigma \vdash M : t}{\Gamma, \Sigma \vdash \text{ref } M : \text{Ref } t} \qquad \mathbf{TDEREF} : \frac{\Gamma, \Sigma \vdash M : \text{Ref } t}{\Gamma, \Sigma \vdash !M : t} \\
 \\
 \mathbf{TASSIGN} : \frac{\Gamma, \Sigma \vdash M_1 : \text{Ref } t \quad \Gamma, \Sigma \vdash M_2 : t}{\Gamma, \Sigma \vdash M_1 := M_2 : \text{Unit}}
 \end{array}$$

Le barème était de (0,7) par règle.  
 La question a été assez bien abordée par ceux qui l'ont faite. L'erreur la plus fréquente concerne la règle TLOC. En effet, le type de l'adresse dans  $\Sigma$  est celui du contenu de la référence, il faut donc ajouter un *Ref* au type récupéré.  
 Il y avait une typo dans l'énoncé puisque l'on souhaitait typer l'affectation (et pas l'appel). Cela n'a posé de problème à presque personne, ceux qui ont donné une règle correcte d'application ont obtenu tous les points.

### Exercice 3 (Un petit méta-modèle objet) [36 mn – 5,9/20]

On souhaite construire un méta-modèle objet simplifié pour pouvoir construire des modèles objets simples comme celui de la figure 1. Ce méta-modèle permet de décrire des *classes* qui possèdent des *attributs* typés, et des *méthodes*. Les méthodes peuvent avoir plusieurs *paramètres* typés et un type de retour. Chaque classe *hérite* d'au plus une classe. Il peut y avoir des *associations* entre classes *sans* notion de multiplicité.

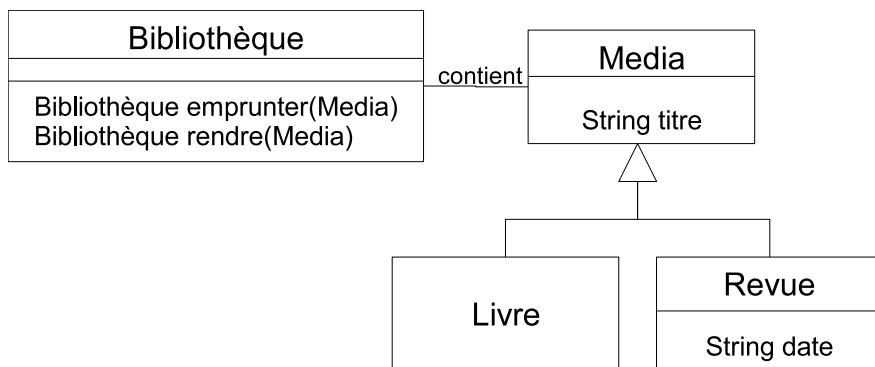


FIG. 1 – Le modèle à construire

**Question 1****[20 mn – 3,3/20]**

- ▷ **Décrire un premier méta-modèle MMOS-1 en s'appuyant sur une notation à la UML<sup>2</sup>. Vous utiliserez un diagramme de classe qui montre toutes les méta-classes et méta-relations nécessaires.**

Il faut décrire un modèle UML qui fait apparaître les classes "Classe", "Methode", "Attribut", "Association", "Paramètre" et les relations entre elles. Les cardinalités par défaut sont 1. Il est évidemment aussi important de les décrire. Par exemple, une classe peut posséder plusieurs attributs (0..\*) ou plusieurs méthodes (\*).

La notion "héríte" est représentée de manière simple par une relation réflexive sur la classe "Classe", avec les bonnes cardinalités pour de l'héritage simple (au plus une parente).

**Question 2****[1 mn – 0,2/20]**

- ▷ **Quelle est la nature d'une instance de la méta-classe Classe ?**

C'est une classe. La méta-classe Classe permet de définir des objets de nature "Classe".

**Question 3****[3 mn – 0,5/20]**

Dans certaines méthodes d'analyse comme celle de Booch, on commence par dresser une liste des attributs avant toute identification de classe.

- ▷ **Le modèle que *vous* avez décrit dans la question 1 permet-il de construire un modèle avec des attributs qui ne sont pas attachés à une classe ?**

La réponse dépend de la cardinalité effective dans votre modèle entre les classes "Classe" et "Attribut". Si un attribut pouvait exister sans être rattaché à une classe (0..) la réponse était OUI sinon (1, 1..) la réponse était NON.

**Question 4****[5 mn – 0,8/20]**

Vous avez à votre disposition tous les constructeurs et accesseurs (lecture écriture des méta-attributs, ajout et retrait dans des collections d'objets) induits naturellement par MMOS-1.

- ▷ **Décrire par un programme (pseudo-code) l'ensemble des opérations nécessaires pour construire<sup>3</sup> le modèle objet de la figure 1.**

Il fallait déclarer les objets (ici les classes) à créer, les créer puis les assembler... Ce qui donne en pseudo code :

<sup>2</sup>La figure 1 met en évidence les éléments syntaxiques d'UML dont vous aurez besoin : boîte, lien entre boîte, etc. En UML faites toutefois attention à la notion de multiplicité qui n'existe pas dans MMOS-1. Si besoin utilisez des commentaires pour exprimer en français (ou en anglais) ce que vous n'arrivez pas à décrire en UML.

<sup>3</sup>Pas graphiquement ! En assemblant les instances du méta-modèle. . .

```
Classe bibliotheque := new Classe("Bibliothèque"); // vous
disposez des constructeurs que vous souhaitez !
Classe media := new Classe("Media");
bibliotheque.addRelation(media,"contient");
bibliotheque.addMethode(new Methode("emprunter", {media}));
etc.
```

Je n'ai pas exigé que tout le modèle soit décrit, mais que l'idée de construction ci-dessus soit explicite.

### Question 5

[3 mn – 0,5/20]

▷ Peut-on instancier une instance de la méta-classe Classe ? Qu'en pensez-vous ?

On imagine bien qu'une instance de l'instance de la meta-classe Classe pourrait exister, mais notre meta-modèle ne définit pas la notion d'instance. Dès lors comment en créer une ? La réponse est NON.

Il faudrait pouvoir écrire : `bibliothequeinstancie()` ;. Mais quelle est le type de l'objet retourné ? Pas ce "concept" dans le méta-modèle !

### Question 6

[2 mn – 0,3/20]

On étend MMOS-1 et on fabrique MMOS-2 en ajoutant la notion d'instance à MMOS-1.

▷ Compléter le diagramme de la question 1.

Il suffit de créer une meta-classe "Instance" reliée à "Classe" (Il y a des "détails techniques" pour attribuer une valeur aux Attributs, mais il était inutile de s'y attaquer.)

### Question 7

[2 mn – 0,3/20]

▷ Peut-on instancier une instance de la méta-classe Classe dans MMOS-2 ? Expliquer votre réponse.

Maintenant, il est possible d'écrire : `Instance biblio_univ := bibliothequeinstancie();`

### Exercice 4 (*Un $\pi$ -calcul un peu spécial ...*)

[17 mn – 2,8/20]

Compte tenu des erreurs de l'énoncé et du fait que la correction n'est pas parvenu correctement à Rennes (même si cela ne perturbait le reste de l'exercice), nous avons choisi de faire basculer cet exercice hors barème.

Cet exercice était considéré comme une partie difficile de l'examen, les perturbations l'ont rendu sans doute encore plus difficile.

Soit le calcul  $\pi_\delta$ , une extension du  $\pi$ -calcul, défini par la syntaxe et la sémantique étiquetée décrite ci-dessous. Les parties ajoutées sont encadrées.

**Syntaxe**

- Les noms :  $a, \dots$
- Les processus :  $P$

$P ::=$	$0$	Rien
	$  \alpha.P$	Préfixe : actions
	$  P_1 + P_2$	Choix non déterministe
	$  P_1   P_2$	Composition parallèle
	$  \nu aP$	Restriction : création de nom
	$  !P$	Réplication

- Les actions :  $\alpha$

$\alpha ::=$	$a(a')$	Réception d'un nom
	$  \bar{a}a'$	Envoi d'un nom
	$  \delta(a, a')$	?
	$  \tau$	Action silencieuse (interne)

**Sémantique**

$?: \delta(x, y).P \xrightarrow{\bar{x}y} \delta(x, y).0 \mid P$	SEND : $\bar{x}y.P \xrightarrow{\bar{x}y} P$	REC : $x(y).P \xrightarrow{x(y)} P$	TAU : $\tau.P \xrightarrow{\tau} P$
CONG : $\frac{P_1 \equiv P'_1 \quad P'_1 \xrightarrow{\alpha} P'_2 \quad P'_2 \equiv P_2}{P_1 \xrightarrow{\alpha} P_2}$	SUM : $\frac{P_1 \xrightarrow{\alpha} P'_1}{P_1 + P_2 \xrightarrow{\alpha} P'_1}$	RES : $\frac{P \xrightarrow{\alpha} P' \quad a \notin n(\alpha)}{\nu aP \xrightarrow{\alpha} \nu aP'}$	
PAR : $\frac{P_1 \xrightarrow{\alpha} P'_1 \quad bn(\alpha) \cap fn(P_2) = \emptyset}{P_1   P_2 \xrightarrow{\alpha} P'_1   P_2}$	COMM : $\frac{P_1 \xrightarrow{a(x)} P'_1 \quad P_2 \xrightarrow{\bar{a}y} P'_2}{P_1   P_2 \xrightarrow{\tau} [x \mapsto y]P'_1   P'_2}$		
OPEN : $\frac{P \xrightarrow{\bar{a}y} P' \quad y \neq a}{\nu yP \xrightarrow{\bar{a}(y)} P'}$	CLOSE : $\frac{P_1 \xrightarrow{a(x)} P'_1 \quad P_2 \xrightarrow{\bar{a}(y)} P'_2 \quad y \notin fn(P_1)}{P_1   P_2 \xrightarrow{\tau} \nu y([x \mapsto y]P'_1   P'_2)}$		

**Question 1**

[15 mn – 2,5/20]

▷ Comment le terme ci-dessous peut-il se réduire ? N'hésitez pas à simplifier l'arbre de dérivation, en réalisant plusieurs réductions simultanément.

$$P = \delta(x, y).!y \mid x(t).\bar{t} \mid x(u).\bar{u}$$

Voici une suite de réductions possible :

On a  $\delta(x, y).!y \xrightarrow{\bar{x}y} \delta(x, y).0 \mid !y$  et  $x(t).\bar{t} \xrightarrow{x(t)} \bar{t}$ , donc par COMM :

$$\delta(x, y).!y \mid x(t).\bar{t} \xrightarrow{\tau} \delta(x, y).0 \mid !y \mid [t \mapsto y]\bar{t} \equiv \delta(x, y).0 \mid !y \mid \bar{y}$$

$!y \equiv y.!y \xrightarrow{y} !y$  et  $\bar{y} \xrightarrow{\bar{y}} 0$ , donc par COMM :  $!y \mid \bar{y} \xrightarrow{\tau} !y \mid 0 \equiv !y$

On a donc :

$$\delta(x, y).!y \mid x(t).\bar{t} \mid x(u).\bar{u} \xrightarrow{\tau} !y \mid \delta(x, y).0 \mid x(u).\bar{u}$$

Or, comme  $\delta(x, y).0 \xrightarrow{\bar{x}y} \delta(x, y).0 \mid 0 \equiv \delta(x, y).0$  et  $x(u).\bar{u} \xrightarrow{x(u)} \bar{u}$ , par COMM :

$$\delta(x, y).0 \mid x(u).\bar{u} \xrightarrow{\tau} \delta(x, y).0 \mid \bar{y}$$

Donc :

$$\begin{aligned} \delta(x, y).!y \mid x(t).\bar{t} \mid x(u).\bar{u} &\xrightarrow{\tau} !y \mid \bar{y} \mid \delta(x, y).0 \\ &\xrightarrow{\tau} !y \mid \delta(x, y).0 \end{aligned}$$

Un certain nombre de personne se sont arrêtés en cours de réduction mais cela ne gênait pas pour la suite. Le seul but de cette réduction était de vous faire sentir la communication par diffusion asynchrone.

## Question 2

[2 mn – 0,3/20]

▷ Quel nouveau type de communication l'action  $\delta$  introduit-elle ?

À partir de la réduction précédente, on constate que le terme obtenu est « équivalent » (par réduction . . .) à  $\delta(x, y).!y$ . Donc cet envoi se diffuse puis reste dans l'environnement.

$\delta$  est donc une opération de diffusion asynchrone.

En fait, on peut discuter l'asynchronisme puisque le premier envoi ne peut se faire que lorsqu'un récepteur est disponible. Donc, le lancement de la diffusion est synchrone en revanche le terme est ensuite dans le milieu et donc le reste de la diffusion est asynchrone.

## Question 3

bonus

▷ Trouver un encodage  $\llbracket \cdot \rrbracket$  de  $\pi_\delta$  dans le  $\pi$ -calcul tel que  $\llbracket P \rrbracket$  soit bisimilaire (noté  $\sim$ ) à  $P$ .

$$\begin{aligned} \llbracket \delta(x, y).P \rrbracket &= \bar{x}y.(!\bar{x}y.0 \mid \llbracket P \rrbracket) \\ \llbracket 0 \rrbracket &= 0 \\ \llbracket \alpha.P \rrbracket &= \alpha.\llbracket P \rrbracket \quad \text{pour les autres actions} \\ \llbracket P_1 + P_2 \rrbracket &= \llbracket P_1 \rrbracket + \llbracket P_2 \rrbracket \\ \llbracket P_1 \mid P_2 \rrbracket &= \llbracket P_1 \rrbracket \mid \llbracket P_2 \rrbracket \\ \llbracket \nu a P \rrbracket &= \nu a \llbracket P \rrbracket \\ \llbracket !P \rrbracket &= !\llbracket P \rrbracket \end{aligned}$$

Il faut ensuite montrer que  $\llbracket P \rrbracket \sim P$ . Le seul cas à faire est  $\llbracket \delta(x, y).P \rrbracket \sim \delta(x, y).P$  (en

supposant  $\llbracket P \rrbracket \sim P$ , les autres étant triviaux.

De  $\delta(x, y).P$  et de  $\llbracket \delta(x, y).P \rrbracket = \bar{x}y.(!\bar{x}y.0 \mid \llbracket P \rrbracket)$  la seule transition possible est  $\xrightarrow{\bar{x}y}$ . Il suffit donc de montrer que  $!\bar{x}y.0 \mid \llbracket P \rrbracket \sim \delta(x, y).0 \mid P$ . Notons  $P_g$  le membre de gauche et  $P_d$  celui de droite.

Les transitions possibles depuis  $P_g$  sont :

- $\xrightarrow{\bar{x}y} !\bar{x}y.0 \mid \llbracket P \rrbracket$  car  $!\bar{x}y.0 \equiv \bar{x}y.!\bar{x}y.0$ , cette transition est possible depuis  $P_d$  et mène à  $\xrightarrow{\bar{x}y} \delta(x, y).0 \mid 0 \mid P$  qui est équivalent à  $\delta(x, y).0 \mid P$ . Ils sont donc bien similaires (couple  $P_g, P_d$ ).
- une transition étiquetée  $\alpha$  de  $\llbracket P \rrbracket$  (si il en existe une), comme par hypothèse  $\llbracket P \rrbracket \sim P$ , toute transition de  $\llbracket P \rrbracket$  étiquetée  $\alpha$  est possible depuis  $P$  (et donc depuis  $\delta(x, y).0 \mid P$ ) et mène à un état bisimilaire (les couples sont donc ceux d'une bisimulation quelconque entre  $\llbracket P \rrbracket$  et  $P$ ).

Il est alors aisé de s'assurer que la relation ainsi construite (qui est une simulation de gauche vers droite) est également une simulation dans l'autre sens.