Licence Info **Corrigé**

LI5a: Développement de programmes

(A. Slissenko)

Examen du 08 Juin 2004, 14h–18h

1 [2 points]. Quelles considérations prend-t-on en compte pour construire un jeu de tests?

Trouver un jeu de tests qui couvre le programme suivant (le symbole ";" signifie la composition séquentielle):

If $|x| = 2 \wedge |y| = 3$ **Then Return** 0 **EndIf**;
If $|x| = 2$ **Then Return** $(y^2 - 9)$ **EndIf**;
$z := x^2$; $u := (z-4)(x+2) - (z - y^2 + 5)$; **Return** u

Quelle fonction calcule ce programme ?

Réponse à 1:

Pour construire un jeu de tests on prend en compte des considérations sémantiques fondées sur notre compréhension du problème, et des considérations fondées sur la structure du programme. Les premières nous disent qu'il faut avoir des tests qui vérifient les cas typiques et les cas marginaux de l'ensemble de situations; les deuxièmes nous disent que nos tests doivent couvrir le programme et doivent exécuter les branches du programme les plus importantes du point de vue des spécifications.

Pour le programme donné un jeu de tests qui le couvre est le suivant: $(x, y) = (2, 3), (2, 0), (0, 0)$.

Le programme calcule la fonction $f(x, y) = (x^2 - 4)(x + 2) - (x^2 - y^2 + 5)$.

****

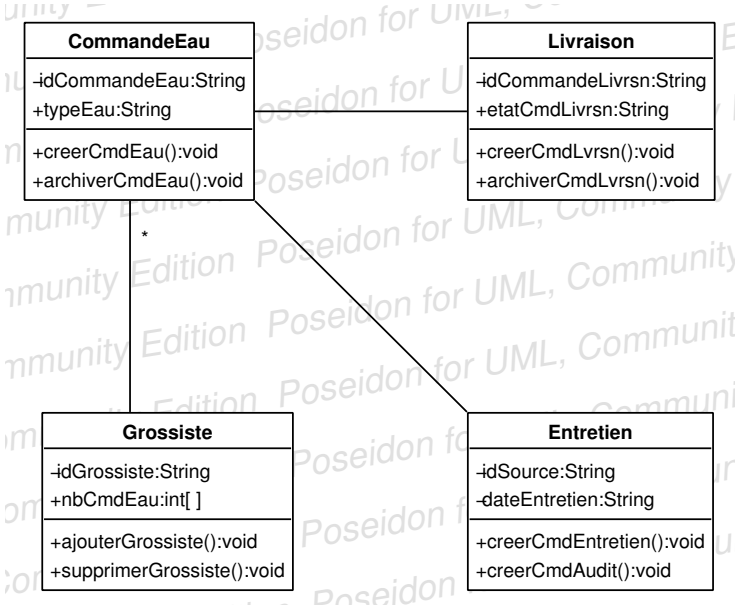
2 [6 points]. Décrire un diagramme de cas d'utilisation et un diagramme de classes pour un programme qui gère les ventes d'eaux minérales. Il y a plusieurs sources qui fournissent des eaux de types différents (on n'exclut pas le cas où un type d'eau est fourni par plusieurs sources, mais le type d'une source est toujours le même). Une source peut être capable de produire une quantité donnée d'eau ou non, cela dépend de sa capacité. Les commandes d'eau son faites par des grossistes qui demandent une quantité d'eau d'un certain type. Le programme doit traiter les commandes et gérer la livraison qui a aussi des capacités limitées. De plus le programme doit planifier l'entretien systématique des sources.

Votre diagramme de classes doit avoir 4 classes et pour chaque classe — 2 attributs et 2 opérations. Choisissez les classes, atributs et opérations les plus importants de votre point de vue. Pour le diagramme de cas d'utilisation choisissez 4 cas d'utilisation.

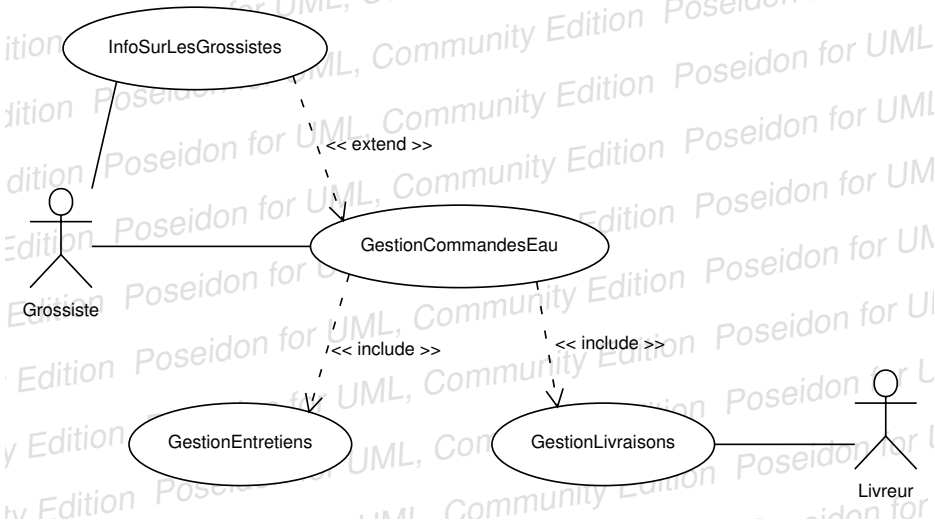
Réponse à 2:

Le texte indique que le programme doit gérer les commandes de grossistes, les livraisons et l'entretien de sources. Donc, on peut introduire les classes: CommandeEau, Livraison, Entretien. Comme 4ème classe on peut prendre soit Grossiste, soit Livreur, soit Source.

Un diagramme de classes possible:



Un diagramme de cas d'utilisation possible:



****

3 [6 points]. Décrire une ASM en utilisant seulement des **If-Then**-opérateurs exécutés en parallèle pour contrôler le travail suivant de lancement de tâches sur N processeurs. Votre ASM doit gérer N processeurs dont chacun peut être soit occupé soit libre. Cet état est géré par la machine à l'aide d'une fonction $Free$ (initialement tous les processeurs sont libres). La machine reçoit des tâches à lancer via N entrées $Task$ correspondant aux processeurs, la valeur de chacune de ces entrées est soit $undef$ soit un objet à traiter (par exemple un objet peut être une instance d'un problème d'optimisation); $undef$ veut dire qu'il n'y a pas de tâche. Ayant reçu une tâche sur une certaine entrée, la machine l'envoie au processeur si il est libre, sinon elle attend et ne regarde pas les autres tâches de cette entrée. Pour lancer une tâche la machine affecte la valeur $vrai$ à la fonction $Launch$ pour cette tâche. Lorsque le processeur termine la tâche il retourne la valeur $vrai$ pour la fonction $Done$ qui peut être utilisée par l'ASM comme une entrée.

Décrire un diagramme Statechart (Etats-Transitions) pour 2 processeurs.

Réponse à 3:

Sortes:

- $\mathbb{P} =_{df} \{1, \ldots, N\}$ (processeurs)
- $TaskInstance$ (une sorte abstraite pour les tâches)

Fonctions:

- $Task : \mathbb{P} \to TaskInstance \cup \{undef\}$ (entrée)
- $Done : \mathbb{P} \to Bool$ (entrée)
- $Free : \mathbb{P} \to Bool$ (sortie)
- $Launch : TaskInstance \times \mathbb{P} \to Bool$ (sortie; $Launch(Task(x), x)$ signifie qu'on lance la tâche $Task(x)$ sur le processeur x)
- $\alpha : \mathbb{P} \to TaskInstance$ (internal function qui sauvegarde la valeur courante de $Task$)

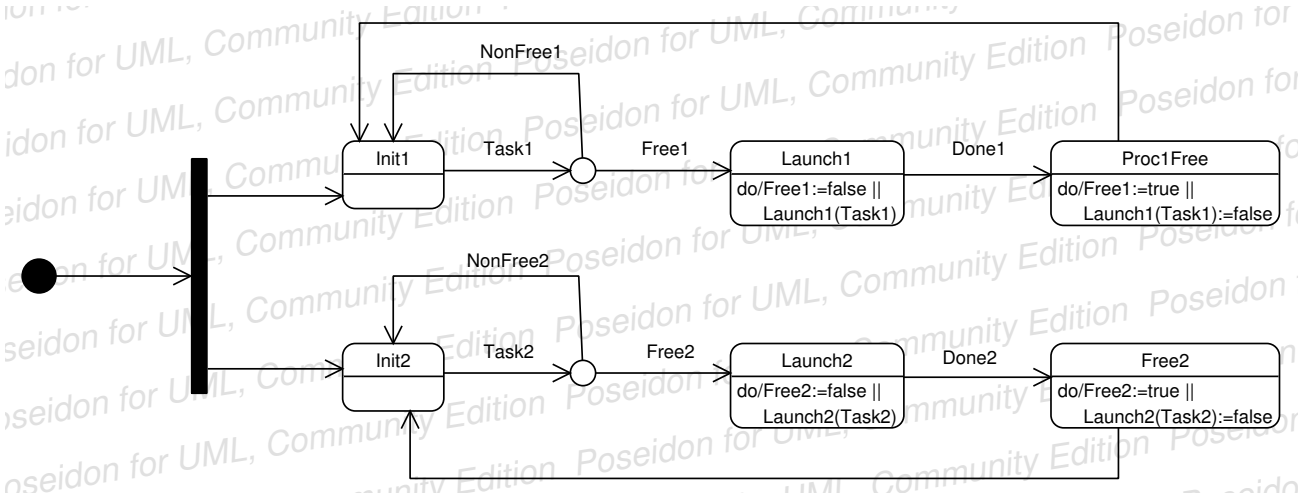
ASM:

Initialisation: $Task(x) = undef$, $Free(x)$ pour tout $x \in \mathbb{P}$.

ForAll $x \in \mathbb{P}$ **Do**
If $Task(x) \neq undef \wedge Free(x)$ **Then** $[Launch(Task(x), x) := true,\ Free(x) := false,$
$\alpha(x) := Task(x)]$
If $Done(x)$ **Then** $[Free(x) := true,\ Launch(\alpha(x), x) := false]$

ici [... , ... , ...] est composition parallèle (la notation du cours).

Diagramme d'états pour deux processeurs:



Ici $FreeJ$, $NonFreeJ$ ($J = 1, 2$) sur les flêches sont des gardes; $TaskJ$ et $DoneJ$ sont des entrées "trigger". Ici on n'utilise pas α car la valeur de $TaskJ$ est la valeur saisie sur la flêche.

****

4 [6 points]. Décrire une ASM (en utilisant seulement des **If-Then**-opérateurs exécutés en parallèle) qui effectue le changement des couleurs de N feux (fonction de sortie Feu) qui sont numérotés $\{0, 1, \ldots, N-1\}$. Chaque feu peut être soit *vert* soit *jaune* soit *rouge*. La machine reçoit un seul signal *Switch* avec les valeurs $undef$, *vert*, *rouge*, *jaune*. Si le signal est *jaune* alors la machine cherche le premier (avec le plus petit numéro) feu vert et le passe au *jaune*. S'il n'y en a pas, alors elle cherche le premier feu rouge et le passe au jaune. Sinon elle ne fait rien. Le signal *vert* passe le premier *rouge* au vert; s'il n'y en a pas – elle ne fait rien. Pareil pour le *rouge* qui change le premier *vert* en *jaune*, sinon – rien.

Réponse à 4:

Sortes:

- $Feux = \{0, 1, \ldots, N-1\}$; cette sorte est considérée comme sous-sorte de $\mathbb{N}$. Donc on peut utiliser les relations et fonctions mathématiques standards pour les nombres naturels, par example, min.
- $Couleurs = \{vert, rouge, jaune\}$

Fonctions:

- $Switch :\rightarrow Couleurs \cup \{undef\}$ (entrée)
- $Feu : Feux \rightarrow Couleurs$ (sortie)

ASM:

Initialisation: $Switch = undef$, $Feu(x)$ est arbitraire pour $x \in \mathbb{P}$.

If $Switch = jaune \wedge \exists\, x\, Feu(x) = vert$ **Then** $Feu(\min\{x : Feu(x) = vert\}) := jaune$
If $Switch = jaune \wedge \neg \exists\, x\, Feu(x) = vert \wedge \exists\, x\, Feu(x) = rouge$
Then $Feu(\min\{x : Feu(x) = rouge\}) := jaune$
If $Switch = vert \wedge \exists\, x\, Feu(x) = rouge$ **Then** $Feu(\min\{x : Feu(x) = rouge\}) := vert$
If $Switch = rouge \wedge \exists\, x\, Feu(x) = vert$ **Then** $Feu(\min\{x : Feu(x) = vert\}) := jaune$

On suppose (par défaut) que Switch reste inchangé suffisamment longtemps pour que la machine ci-dessus puisse exécuter une fois un opérateur **If-Then**.
Si vous voulez expliciter la recherche des numéros $\min\{x : Feu(x) = vert\}$, $\min\{x : Feu(x) = rouge\}$ if faut imposer une contrainte plus forte: *Switch* ne change pas pendant cette recherche. Mais la spécification ne demande pas de détailler cette recherche.

****