Corrigé

Master 1 Informatique 2006–2007

ASILANG0 : Langages de spécification et de développement de logiciels

Examen

Le 10 janvier 2007, 14h–16h.
Utilisation des notes, livres, docs, etc. autorisée.

1. [14 points] Considérons le diagramme SDL suivant. Il consiste en 2 paquetages `predefined` et `NourrirAnmlSignals` et en un bloc `NourrirController`. Le paquetage `NourrirAnmlSignals` est le suivant :

```
signal
nourrir(Boolean),
auge(augeState) ;
signal
alarme(Boolean) ;
signal
openAuge(Boolean) ;

newtype augeState
LITERALS
vide,prise,libre ;
endNewType ;
```

Le bloc `NourrirController` défini sur la Fig. 1 consiste en un seul processus `nourrirProcess` représenté sur la Fig. 2. Ce processus gère la nutrition d'un animal. Le signal `nourrir` à valeurs booléennes commande une séance de nutrition (la valeur de `nourrir` égale à `true` dit qu'il faut nourrir l'animal). La fin de la séance est commandée par la valeur `false` du même signal. Pour commencer la séance le contrôleur doit faire le signal `openAuge` si l'auge est `libre`. Sinon il doit déclencher une alarme qui dure jusqu'au moment où l'auge devient disponible. Dans ce cas l'alarme doit être arrêtée et l'auge doit être ouverte.

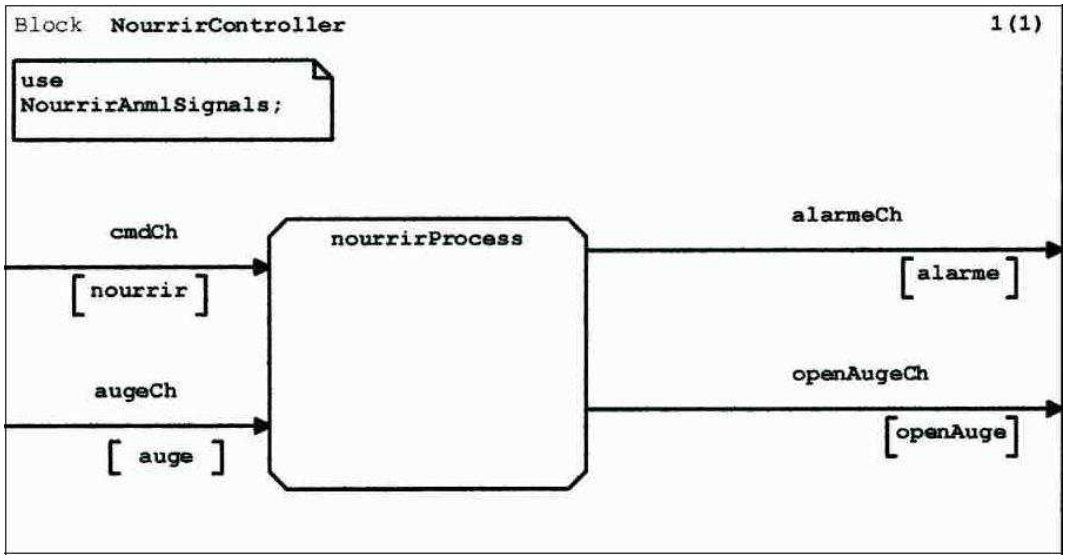

FIG. 1 – SDL bloc `NourrirController`

Répondez aux questions (a)–(d) :

(a) Quels types du paquetage `predefined` sont-ils utilisés dans les diagrammes ?

(b) Quel est le rôle des variables x, y, z ?

(c) Le type de la variable y peut-il être simplifié ?

(d) Quel est le rôle du texte `use NourrirAnmlSignals` dans le diagramme du bloc ?

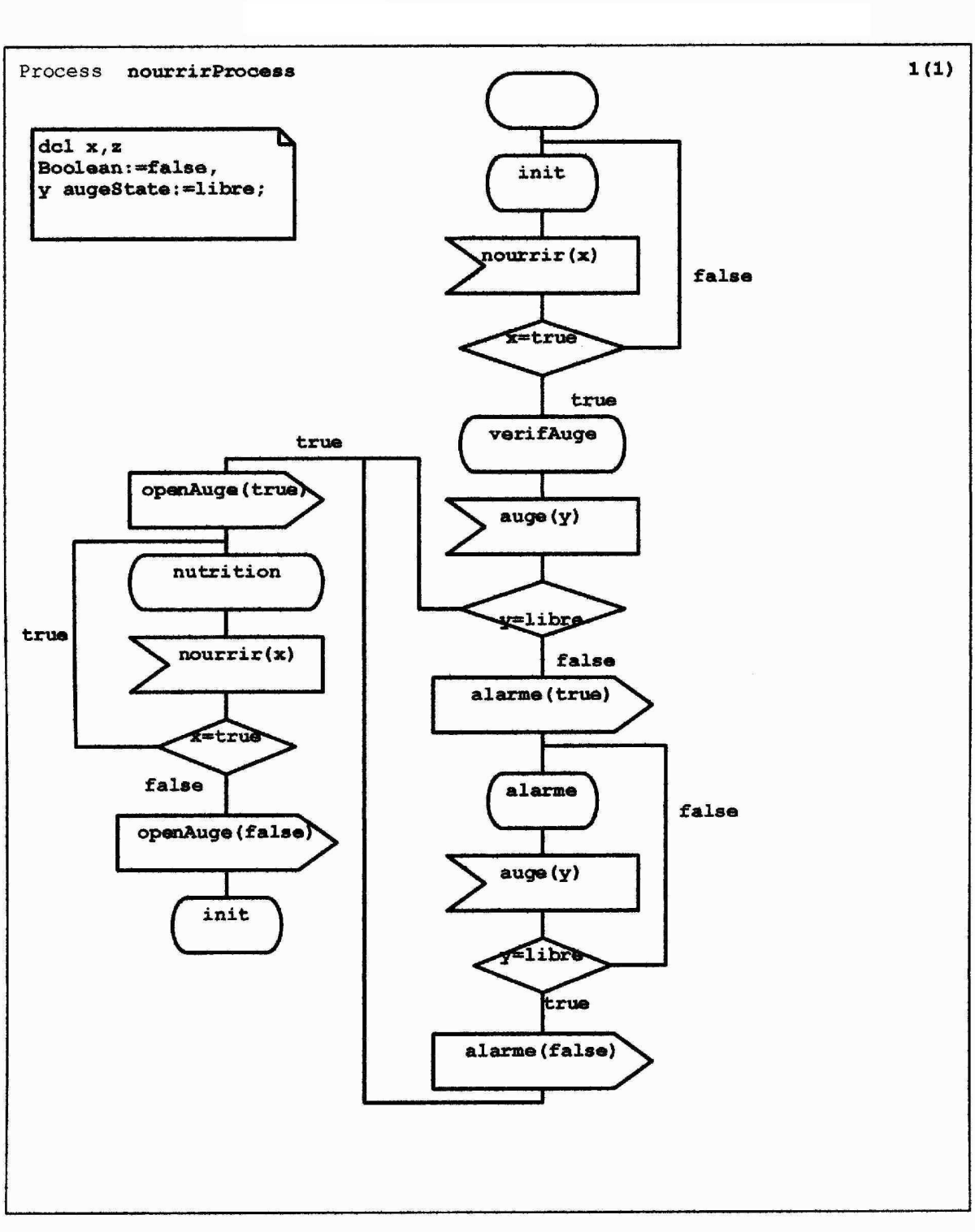

FIG. 2 – SDL processus `nourrirProcess`

Décrivez une ASM pour le même contrôleur. Faites votre ASM à un niveau d'abstraction plus élevée que celui du processus `nourrirProcess`. Plus concrètement, n'utilisez pas les variables telle que x pour sauvegarder les valeurs des signaux. Utilisez les fonctions qui représentent les signaux directement. N'introduisez pas les états de `nourrirProcess` explicitement. Utilisez les signaux de `nourrirProcess`, mais prenez en compte que la syntaxe d'utilisation de fonctions dans les diagrammes SDL est différente de celle des ASM.

Réponse.

Réponses aux questions :

(a) Boolean.

(b) x sauvegarde la valeur de *nourrir*, y sauvegarde la valeur de *auge*, z n'est pas utilisée (donc elle sert à rien).

(c) y peut être traitée comme variable booléenne, dans ce cas il faut changer le type de la fonction *auge* en $auge(Boolean)$ dans le paquetage `NourrirAnmlSignals` et respectivement l'expression $y = libre$ dans le lozenge de décision dans le diagramme du processus `nourrirProcess` en $y = true$.

(d) Le texte `use NourrirAnmlSignals` permet de référencer (ou faire visibles) les signaux spécifies dans le paquetage `NourrirAnmlSignals` dans le bloc `NourrirController`, en particulier dans le processus `nourrirProcess`.

ASMs qui modélisent le diagramme SDL.

Vocabulaire. Remarquons que nous n'utilisons que les sortes par défaut.

Fonctions.

- $nourrir :\rightarrow Bool : false$ (entrée, "$false$" après "$Bool$:" donne la valeur initiale)
- $auge :\rightarrow Bool : true$ (entrée, $auge$ correspond à $auge = libre$ du diagramme)
- $openAuge :\rightarrow Bool : false$ (sortie)
- $alarme :\rightarrow Bool : false$ (sortie)

Remarquons que selon le diagramme du processus `nourrirProcess` le contrôleur ouvre l'auge dans le cas où l'alarme est déclenchée et l'auge devient disponible même si le signal *nourrir* est devenu déjà faux. Par ailleurs, dans ce cas le contrôleur ferme l'auge tout de suite après.

Programme (ASM de base)

```
1 : if nourrir ∧ auge ∧ ¬ openAuge then openAuge := true
2 : if ¬ nourrir ∧ openAuge then openAuge := false
3 : if nourrir ∧ ¬ auge ∧ ¬ alarme then alarme := true
4 : if alarme ∧ auge then [ alarme := false, openAuge := true ]
```

Une Une tentative d'écrire une ASM avec la composition séquentielle.

```
1 : if nourrir
      then
2 :   if ¬ auge
        then
3 :     ( alarme := true ;
4 :       while ¬ auge do skip ;
5 :       alarme := false ;
6 :       openAuge := true )
        else
7 :     ( openAuge := true ;
8 :       while nourrir do skip ;
9 :       openAuge := false )
```

Ce programme modélise le diagramme SDL mais ne le suit pas lettre à lettre. Par exemple, si on est dans la branche **if** $\neg auge$ **then**... et l'auge devient disponible, alors on ouvre l'auge et on sort de la boucle externe. Ensuite on vérifie si la fonction *nourrir* est vraie (comme dans le diagramme) et ensuite, si c'est le cas, on vérifie si l'auge est disponible. Cette dernière vérification ne correspond pas au diagramme. Si on suppose que la disponibilité de l'auge ne change pas durant le temps où $nourrir = true$, alors ce programme peut être considéré adéquate.

Voilà une ASM qui suit le diagramme précisément :

```
1 : if nourrir
      then
2 :   if auge
3 :     then ( openAuge := true ; while nourrir do skip ; openAuge := false )
4 :     else ( alarme := true ; while ¬ auge do skip ; alarme := false ; openAuge := true ;
                 while nourrir do skip ; openAuge := false )
```

On voit que ce dernier programme contient 2 morceaux identiques qui correspondent au sous-

diagramme gauche. Donc, pour faire l'ASM plus compacte on peut introduire la règle :

$Rnourrir = \big(\, openAuge := true\,;\ \textbf{while}\ nourrir\ \textbf{do skip}\,;\ openAuge := false\,\big)$

Avec cette règle la dernière ASM devient :

```
1 :   if nourrir
        then
2 :     if auge
3 :       then Rnourrir
4 :       else ( alarme := true ;  while ¬ auge do skip ; alarme := false ;  Rnourrir )
```

Les meilleures réponses parmi les trois ci-dessus sont la 1ère et la 3ème machines.

2. [14 points] Maintenant il s'agit d'écrire une ASM qui contrôle la nutrition de plusieurs animaux. La différence principale avec le sujet **1** est que cette fois la durée d'une séance de nutrition est définie par une contrainte temporelle. Si un signal de nourrir un animal est déclenché, alors cet animal doit être nourri pendant une durée donnée, peut-être avec des changements d'auges et des pauses s'il n'y a pas d'auges disponibles.

Les animaux constituent une sorte *Animal* et les auges constituent une sorte *Auge*. Les auges sont représentées par des nombres naturels, donc on pourra parler de l'auge disponible à un moment donné ayant le plus petit numéro. Pour ne pas confondre les notations de SDL et d'ASM on donne aux fonctions du sujet 2 des noms différents du sujet 1 précédent. Comme variables logiques pour les éléments d'*Animal* utilisez la lettre α (avec des indices s'il vous faut plusieurs variables), et comme variables logiques pour les éléments d'*Auge* utilisez la lettre η (aussi avec des indices si nécessaire).

Le signal d'entrée qui commande de nourrir un animal est *nrr*. Il dépend de l'animal et est à valeurs booléennes (*true* dit qu'il faut nourrir l'animal). À un moment donné ce signal ne peut être déclenché que pour un seul animal. Bien sûr, pendant qu'il reste vrai pour un animal donné il peut être déclenché pour un autre animal. Cependant les moments de la fin peuvent coïncider pour plusieurs animaux. De plus, on suppose qu'au moment de déclenchement de ce signal aucune auge ne change son état et on suppose que dans toutes les situations au plus un seul animal peut être en attente d'une auge.

L'état d'une auge est donné par la fonction *aug* à valeurs $\{libre, nonLibre\}$. La durée pour nourrir un animal est donnée par la fonction statique Δ à valeurs réelles positives.

Si le signal *nrr* est déclenché pour un animal α, le contrôleur regarde s'il y a une auge disponible (ce qui correspond à la valeur *libre*) et s'il y en aune au moins, il ouvre la auge disponible ayant le plus petit numéro en affectant à la fonction *ouvr* la valeur *true*. Bien évidemment, cette spécification présume que *ouvr* est à valeurs booléennes et qu'elle a comme argument une auge et un animal.

Dans ce sujet il n'y a pas d'alarme. Le signal $nrr(\alpha)$ reste vrai pendant un certain temps qui ne dépasse pas $\Delta(\alpha)$. Néanmoins, si $nrr(\alpha)$ devient $false$ mais la durée $\Delta(\alpha)$ de nutrition de α n'a pas été assurée, alors il faut continuer à nourrir cet animal. Cependant pendant une séance de nutrition l'auge utilisée peut devenir *nonLibre* (par exemple, cela peut être interprété comme "vide", "cassée" etc.). Dans ce cas il faut fermer l'auge utilisée et en chercher une autre. S'il n'y en a pas, il faut attendre une auge disponible. Intuitivement, on suppose que les attentes sont courtes, ou plus précisément, pendant une séance de nutrition d'un animal le signal *nrr* ne peut pas être déclenché une deuxième fois (i. e. passer de vrai à faux et ensuite encore à vrai).

Par exemple, supposons que $\Delta(\alpha) = 3$ et qu'à un moment t le signal $nrr(\alpha)$ devient vrai et reste vrai jusqu'au moment $(t + 1.5)$. Supposons qu'au moment t il y a une auge disponible. Alors le contrôleur ouvre cette auge pour α. Supposons qu'elle est utilisée jusqu'au moment $(t + 1)$ où aucune auge n'est plus disponible. Donc il reste à nourrir α pendant 2 unités de temps. Ensuite il n'y a pas d'auges disponibles jusqu'au moment $(t + 4.6)$ où il y en a une. Donc le contrôleur ouvre cette nouvelle auge pour α jusqu'au moment $(t + 6.6)$ s'il n'y a pas d'autre interruption de la disponibilité. La spécification de l'environnement nous assure que pendant le temps entre $(t + 1.5)$ et $(t + 6.6)$ le signal $nrr(\alpha)$ reste faux et qu'il n'y a jamais deux animaux en attente d'une auge disponible.

Remarquons que l'ordre de nutrition des animaux est imprévisible. Certains animaux mangent pendant certaines périodes plus souvent, d'autres moins souvent.

Voici quelques indications qui peuvent vous aider à développer une ASM. Introduisez les notations : $\varphi(\eta)$ qui dit que l'auge η est libre et n'est utilisée pour aucun animal, et ν qui est égale à la auge disponible ayant le plus petit numéro (vous devez écrire $\varphi(\eta)$ et ν comme formules sur le vocabulaire de votre ASM). Ensuite introduisez les fonctions internes suivantes :

- *dl* qui calcule pour un animal donné le moment où il faut arrêter sa nutrition si tout va bien, i. e. si des auges sont disponibles sans interruption. Les valeurs de cette fonction sont du type $Temps \cup \{\infty\}$, cette valeur ∞ permet d'utiliser *dl* comme drapeau.
- δ à valeurs du type $Temps$ qui donne le temps restant de nutrition dans le cas d'interruption. Initialement et au début de chaque séance de nutrition δ est égale à Δ.
- μ à valeurs du type $Auge \cup Undef$ qui mémorise l'auge utilisée pour un animal qui est en cours

de nutrition. La valeur initiale peut être $undef$ pout tout animal.
- *séance* à valeurs booléennes qui indique (par la valeur $true$) pour un animal donné si une séance nutrition de cet animal est en cours.

Dans les gardes utilisez les quantificateurs, par exemple, pour dire qu'il y a une auge disponible on dit " il existe une auge η qui vérifie $\varphi(\eta)$ ".

Utilisez le paramètre α dans **forall** de votre ASM. Le signal $nrr(\alpha)$ peut être utilisé pour mettre $séance(\alpha)$ à $true$. Cela peut être la seule utilisation de ce signal de déclenchement d'une séance de nutrition. Si $CT < dl(\alpha) < \infty$ alors le temps qui reste pour nourrir α est évidemment $(dl(\alpha) - CT)$. Donc si à ce moment-là (i. e. CT) la séance de nutrition est interrompue (parce qu'il n'y a pas d'auge disponible du tout), alors le contrôleur mémorise cette différence dans $\delta(\alpha)$.

En gros votre ASM peut fonctionner de la façon suivante. Pour tout animal α elle exécute en parallèle les actions suivantes.

Déclenchement d'une séance de nutrition comme dit ci-dessus.

Si une séance est déclenchée et dl est à sa valeur initiale et il y a une auge disponible, alors on ouvre une auge disponible pour.

Si durant une séance, avant sa fin (on voit cela en comparant la valeur de $dl(\alpha)$ et le temps courant) l'auge utilisée devient non disponible on la ferme et ensuite on agit selon la situation. Le premier cas : il y en a une autre, on ferme l'auge utilisée et on en ouvre une autre. Sinon, c'est le deuxième cas, on mémorise le temps restant (voir ci-dessus) et on attend le premier moment où une auge devient disponible.

Et finalement, si la séance arrive à sa fin (le temps courant atteint $dl(\alpha)$), on initialise les fonctions pertinentes.

Réponse. En fait, une ASM qui donne une réponse, est décrite à la fin du texte ci-dessus.

Vocabulaire.

Sortes. $Animal$, $Auge$, $ValAuge$ (c'est une sorte des valeurs de la fonction aug, il consiste en 2 éléments spécifiés ci-dessous comme constantes $libre$, $nonbLibre$).

Rappelons que $Temps$ est une sorte par défaut.

Fonctions et leur initialisation.

- $nrr : Animal \to Bool : false$ (entrée)
- $aug : Auge \to ValAuge : libre$ (entrée)
- $ouvr : Auge \times Animal \to Bool : false$ (sortie)
- $\Delta : Animal \to Temps$ (statique)
- $dl : Animal \to (Temps \cup \{\infty\}) : \infty$ (interne ; strictement dit, il nous faut définir la sorte $\{\infty\}$)
- $\delta : Animal \to Temps : \forall\, \alpha\, \delta(\alpha) = \Delta(\alpha)$ (interne)
- $\mu : Animal \to (Auge \cup Undef) : undef$
- $séance : Animal \to Bool : false$
- $libre :\to ValAuge$, $nonLibre :\to ValAuge$ (2 constantes du type $ValAuge$)

Notations :

$\varphi(\eta) =_{df} \big(aug(\eta) = libre \wedge \forall\, \alpha \neg\, ouvr(\eta, \alpha) \big)$

$\nu =_{df} \min\{\eta : \varphi(\eta)\}$

Pour que l'ASM ci-dessous soit correcte les contraintes suivantes, formulées dans la spécification, sont essentielles :
- à un moment donné le signal $nourrir$ ne peut être déclenché que pour un seul animal,
- au moment de déclenchement du signal $nourrir$ aucune auge ne change son état,
- dans toutes les situations au plus un seul animal peut être en attente d'une auge.

forall α **do**

1 : **if** $nrr(\alpha) \wedge \neg séance(\alpha)$
then $\left[séance(\alpha) := true \right]$

2 : **if** $séance(\alpha) \wedge dl(\alpha) = \infty \wedge \exists \eta \, \varphi(\eta)$
then $\left[dl(\alpha) := CT + \delta(\alpha),\ \mu(\alpha) := \nu,\ ouvr(\nu, \alpha) := true \right]$

3 : **if** $CT < dl(\alpha) < \infty \wedge aug(\mu(\alpha)) \neq libre \wedge \exists \eta \, \varphi(\eta)$
then $\left[ouvr(\mu(\alpha), \alpha) := false,\ ouvr(\nu, \alpha) := true,\ \mu(\alpha) := \nu \right]$

4 : **if** $CT < dl(\alpha) < \infty \wedge aug(\mu(\alpha)) \neq libre \wedge \neg \exists \eta \, \varphi(\eta)$
then $\left[ouvr(\mu(\alpha), \alpha) := false,\ \delta(\alpha) := dl(\alpha) - CT,\ dl(\alpha) := \infty \right]$

5 : **if** $CT \geq dl(\alpha)$
then $\left[ouvr(\mu(\alpha), \alpha) := false,\ \delta(\alpha) := \Delta(\alpha),\ dl(\alpha) := \infty,\ séance(\alpha) := false \right]$