Vérification et Validation

Année 2019-2020

www.lri.fr/∼wolff/teach-material/2019-20/PolyTech-VnV

Prof. Burkhart Wolff
wolff@lri.fr

Delphine Longuet
longuet@lri.fr

Test fonctionnel

Date : 6 mars 2017

Exercice 1 (Analyse partitionnelle et test aux limites)
Une société vend deux produits A et B au prix unitaire de 5 € pour A et de 10 € pour B. Une commande comprend une certaine quantité du produit A et une certaine quantité du produit B. Le coût d'une commande est la somme totale des prix unitaires des produits commandés, à laquelle on applique une réduction selon les règles suivantes :

- Si la somme totale est supérieure ou égale à 200 €, on applique une réduction de 5%, si elle est supérieure ou égale à 1000 €, la réduction est de 20%. Ces deux réductions ne sont pas cumulables et portent sur la somme totale.
- La société souhaitant encourager la vente de A, on applique, sur le prix obtenu grâce à la règle précédente, une réduction supplémentaire de 10% si la commande comprend au moins 45 produits A.

1. Donnez un ensemble de tests pour le calcul du coût total d'une commande. Pour chaque test :
   - expliquez le cas particulier visé par le test ;
   - donnez la formule du résultat attendu ;
   - donnez un exemple de valeurs concrètes et le résultat correspondant attendu.
2. Complétez votre jeu de tests par une analyse aux limites.

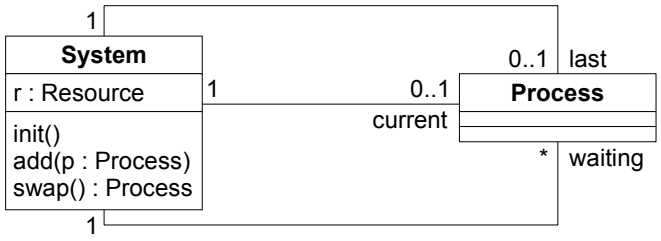
Exercice 2 (Test fonctionnel formel d'une fonction)
L'opération `middle` prend en entrée trois entiers différents deux à deux et renvoie l'entier parmi les trois qui n'est ni le plus grand ni le plus petit.

1. Donnez une spécification formelle de cette opération.
2. Construisez la forme normale disjonctive de cette spécification et déduisez-en un ensemble de cas de test pour l'opération `middle`.
3. Donnez des tests concrets pour chacun des cas trouvés à la question précédente.

Exercice 3 (Test fonctionnel formel d'une méthode)
On considère la spécification suivante.

L'instance de System gère une (unique) ressource, par exemple un processeur, qu'il doit partager entre des processus. Un processus ne termine jamais et ne libère jamais spontanément la ressource mais uniquement sur la requête du système via l'opération swap,

qui permet d'attribuer la ressource à l'un des processus en attente. Le système n'est pas forcément équitable et se contente simplement d'essayer de ne pas redonner la ressource au processus qui le possédait déjà lors de l'échange précédent. Il dispose pour cela de current, l'éventuel processus qui possède actuellement la ressource, de last, l'éventuel processus qui possédait la ressource avant current, et de waiting, l'ensemble des processus demandeurs (qui inclut last).



L'opération init() met le système dans un état initial où il n'existe aucun processus. L'opération add(p :Process) ajoute un processus au système. S'il n'existe aucun processus, p devient le processus courant qui détient la ressource, sinon il est ajouté à waiting. L'opération swap() change le processus actif qui détient la ressource.

$$
\begin{aligned}
inv_{\mathsf{System}}(s) \quad &\equiv \quad s.\mathsf{waiting} \neq \emptyset \longrightarrow s.\mathsf{current} \neq NULL \\
&\quad \wedge\ s.\mathsf{last} \neq NULL \longrightarrow (s.\mathsf{last} \neq s.\mathsf{current} \wedge s.\mathsf{last} \in s.\mathsf{waiting}) \\
&\quad \wedge\ s.\mathsf{current} \neq NULL \longrightarrow s.\mathsf{current} \notin s.\mathsf{waiting}
\end{aligned}
$$

$$
pre_{\mathsf{swap}}()(s) \quad \equiv \quad s.\mathsf{waiting} \neq \emptyset
$$

$$
\begin{aligned}
post_{\mathsf{swap}}(result)(s) \quad &\equiv \quad result \in old(s.\mathsf{waiting}) \wedge s.\mathsf{current} = result \\
&\quad \wedge\ s.\mathsf{waiting} = old(s.\mathsf{waiting}) \setminus \{result\} \cup old(s.\mathsf{current}) \\
&\quad \wedge\ s.\mathsf{last} = old(s.\mathsf{current}) \\
&\quad \wedge\ (old(s.\mathsf{last}) \neq NULL \wedge |old(s.\mathsf{waiting})| > 1 \longrightarrow result \neq old(s.\mathsf{last}))
\end{aligned}
$$

1. Donnez informellement les différents cas de test pour l'opération swap en fonction des valeurs de waiting, last et current avant et après l'opération.
2. Calculez la DNF pour l'opération swap et construisez les cas de test. Commencez le calcul de la DNF avec $inv_{\mathsf{System}}(s)(\sigma_{pre}) \wedge \mathsf{pre}_{\mathsf{swap}}(s)(\sigma_{pre}) \wedge post_{\mathsf{swap}}(s)(\sigma_{pre}, \sigma) \wedge inv_{\mathsf{System}}(s)(\sigma)$.
3. Sélectionnez des tests concrets pour chacun des cas de test déduits à la question précédente, en choisissant des valeurs pour waiting, last et current.
4. Pour chacun des tests obtenus à la question précédente, construisez la suite d'appel des fonctions init, add et swap permettant d'exécuter ce test.