Génie Logiciel. Projet.

Simulateur d'ASMs temporisées

Le but du projet est de développer et d'implémenter un simulateur pour des ASMs temporisées simples dont la forme est décrite ci-dessous. Le contrôleur de passage à niveau pour un nombre concret de voies est un cas particulier de ces machines.

Une machine est donnée par un vocabulaire, un programme et des valeurs initiales. Il n'y a que des sortes finies concrètes et les sortes par défaut : $Bool$, Nil, $Entiers$, $Temps$ (entiers non négatifs). L'utilisateur n'introduit pas les fonctions standards sur ces sortes : les fonctions booléennes ($\wedge$, $\vee$, $\rightarrow$, $\neg$), l'addition, soustraction et multiplication des entiers, les relations d'ordre sur les entiers. Toutes les fonctions abstraites sont d'arité 0.

Un terme est soit une fonction d'arité 0 soit une somme $(a_0 * f_0 + \cdots + a_k * f_k)$, où $*$ dénote la multiplication, k, $a_0, \ldots, a_k$ sont des nombres naturels concrets et $f_0, \ldots, f_k$ sont des fonctions abstraites avec les valeurs entières. Le dernier terme est un *terme arithmétique*.

Une formule atomique est soit un prédicat soit une (in)égalité de deux termes arithmétiques. Une formule est construite à partir des formules atomiques à l'aide des connecteurs booléens.

La syntaxe de programme :

If G_1 **Then** A_1
If G_2 **Then** A_2
. . . . . .
If G_m **Then** A_m

Ici G_i sont des gardes (formules) et A_i sont des listes d'updates interprétées comme composition parallèle. Un update a la forme $f := \theta$, où f est une fonction interne et θ est un terme du type approprié ou une formule logique si le type de f est $Bool$. Une tâche de simulation est définie par une interprétation des fonctions externes.

La syntaxe de représentation d'ASM et de cas de simulation se trouve dans l'annexe.

Le simulateur doit donner la possibilité de simuler le travail d'une machine donnée. Plus précisément il doit assurer les fonctions et propriétés suivantes :

(S1) Le simulateur marche sur au moins deux machines différentes : une machine gère l'interface utilisateur du simulateur et le(s) autre(s) effectue(nt) les tâches de simulation.

(S2) Le simulateur a son horloge dont la vitesse observable peut être changée par l'utilisateur.

(S3) L'interface permet à l'utilisateur de définir une machine et une tâche de simulation directement sur l'écran ou de les charger comme fichiers. Le format de ce fichier doit être le même pour tous les étudiants.

(S4) L'utilisateur a la possibilité de faire un paquet de tâches de simulation.

(S5) Chaque tâche de simulation peut être sauvegardée et peut être *facilement réutilisable*.

(S6) Les résultats de chaque tâche de simulation sont sauvegardés et peuvent être réutilisés.

(S7) L'interface présente dynamiquement l'évolution des états de la machine simulée.

(S8) L'utilisateur peut arrêter le processus de simulation d'une tâche et le reprendre.

(S9) (Option.) Pendant l'arrêt de la simulation d'une tâche l'utilisateur peut changer la partie encore non traitée des entrées et reprendre la simulation avec les données modifiées.

Toutes les étapes de travail doivent être décrites dans le rapport. En particulier, le rapport doit contenir :

(R1) Les requis d'utilisateur.

(R2) La spécification des requis.

(R3) L'architecture du simulateur.

(R4) Les spécifications du simulateur à des niveaux différents.

(R5) Un plan de test et les résultats des tests.

(R6) Des mesures du travail et du logiciel. Les mesures doivent contenir : le volume des programmes écrits, le nombre de fonctions calculées, le nombre de modules et le nombre de versions pour chaque module, le temps dépensé pour écrire un module et le temps pour le valider, les mêmes mesures pour le logiciel entier. Sur le test : la taille du test et sa couverture, le nombre d'erreurs trouvées et le temps moyen pour corriger une erreur. De plus il faut donner les mesures structurelles du logiciel (la taille moyenne d'un module, le nombre de branchements par module, le nombre de boucles, le nombre de boucles imbriquées, le nombre de goto, les dépendances entre les modules).

La programmation doit être faite en Java et en binôme. L'utilisation d'outils est souhaitable et sera appréciée. Le travail en trinôme est admissible, mais il demande d'implémenter l'option mentionnée ci-dessus et une possibilité de lancer deux tâches de simulation en parallèle sur deux machines différentes.

Attention. Si votre programme ne marche pas, votre note sera inférieure à 10/20.

Annexe.

Les sortes et les fonction par défaut sont (vous devez utiliser les notations ci-dessous) :

- Sortes : $Integer$ (Entier), $Temps$ (entiers non négatifs), $Bool$ (valeurs booléennes).
- Constantes (fonctions statiques d'arité zéro) : entiers, $true$, $false$.
- Fonctions : $+$, $-$, $*$, $+(\text{ mod } k)$ (somme modulo k pour un k concret) ; $<$, $>$, $=<$ (inférieur ou égal), $>=$ (supérieur ou égal), $=$, $!=$ (différent) sur les entiers ; *and* (conjonction), *or* (disjonction), *not* (négation) sur les valeurs booléennes ; CT (Current Time) fonction dynamique de type $-> Temps$, interprétée come identité. □

Dans la description du format ci-dessous les mots-clé sont en gras.

Rappelons que les sortes et les fonctions par défaut sont toujours présentes, et donc elles ne sont pas mentionnées dans la description des machines. Le simulateur doit traiter les tâches représentées selon la syntaxe suivante :

Syntaxe.

Machine : $nomASM$.

Sorts : $Sorte1 = \{\beta_{1,1}, \ldots, \beta_{1,s_1}\}$, ... , $SorteK = \{\beta_{K,1}, \ldots, \beta_{K,s_K}\}$.

Constants : /* Ici on liste les constantes de chaque sorte avec le format $a_1, a_2, \ldots, a_\nu : -> Sorte$. */

Inputs : $nomInput_1 : type_1$, ...,$nomInput_{\nu_{Inp}} : type_{\nu_{Inp}}$.

Outputs : $nomOutput_1 : type_1$, ...,$nomOutput_{\nu_{Out}} : type_{\nu_{Out}}$.

Auxiliary : $nomAux1 : type_1$, ...,$nomAux_{\nu_{Aux}} : type_{\nu_{Aux}}$.

/* Les fonctions déclarées dans **Inputs, Outputs, Auxiliary** sont dynamiques et d'arité zéro. */

Program : $Program$.

Initialization :
/* Les valeurs des constantes doivent être choisies parmi les éléments des sortes correspondantes. */

InitConstants : $a_1 = \alpha_1, a_2 = \alpha_2$, ...

/* On peut mettre les parenthèses après le nom de fonction même si celle-ci a pour arité zéro. */

InitOutputs : $nomOut1 = vOut1$, ...,$nomOutM = vOutM$.

InitAux : $nomAux1 = vAux1$, ..., $nomAux\mu = vAux\mu$.

InitInputs : /* L'initialisation d'une fonction appartenant à **Inputs** doit donner une valeur (pour chaque instant de l'intervalle de simulation). */

TimeInterval : $lastMoment$.
/* La simulation va de l'instant 0 jusqu'à l'instant $lastMoment$. */

InputsOrder : $ListOfInputs$.
/* Cette liste ordonne les fonctions.
Cet ordre sera respecté dans les listes de valeurs sur l'intervalle de simulation. */

InputsValues : $ListsOfValues$.

/* Exemple. Si on a 2 entrées $f1 : -> Temps$ et $f2 : -> Temps$ alors $ListOfInputs$

peut être $(f1, f2)$. Supposons que l'intervalle de simulation est $[0, 3]$.
Dans ce cas $ListsOfValues$ sera de la forme $((vf1_0, vf2_0; vf1_1, vf2_1; vf1_2, vf2_2; vf1_3, vf2_3)$.
Par exemple, la valeur $vf2_2$ est la valeur de $f2$ au moment 2.
*/

Un exemple de machine

Machine : ASMExmp1.
Sorts : $Proc = \{0, 1, 2\}$.
Constants : $N : -> Integer$, $d0 : -> Temps$, $d1 : -> Temps$, $p0 : -> Proc$.
Inputs : $Pass : -> Proc$.
Outputs : $Token0 : -> Bool$, $Token1 : -> Bool$, $Token2 : -> Bool$.
Auxiliary : $Last : -> Temps$.

Program :

If$(Pass! = 0$ and $Token1$ and $Pass = 2)$ **Then**
$[Token2 := (d0 =< (CT - Last)$ and $(CT - Last) =< d1)$,
$Token1 := not(d0 =< (CT - Last)$ and $(CT - Last) =< d1)$, $Last := CT]$
If$(Pass! = 0$ and $Token2$ and $Pass = 1)$ **Then**
$[Token1 := (d0 =< (CT - Last)$ and $(CT - Last) =< d1)$,
$Token2 := not(d0 =< (CT - Last)$ and $(CT - Last) =< d1)$, $Last := CT]$

Initialization :

InitConstants : $N = 2, d0 = 2, d1 = 5, p0 = 1$.
InitOutputs : $Token0 = false$, $Token1 = true$, $Token2 = false$.
InitAux : $Last = 0$.
InitInputs :
TimeInterval : 12.
InputsOrder : $(Pass)$.
ListsOfValues : (1 ;0 ;2 ;1 ;0 ;0 ;2 ;1 ;2 ;1 ;2 ;1).

/* Le processus 0 a un rôle spécial, il correspond à la valeur nil. */