Modélisation et vérification des systèmes concurrents

Devoir Maison n° 1

Algorithme de vérification d'un invariant

Soit $ST = (S, Act, \rightarrow, I, Prop, L)$ un système de transition sur un ensemble *Prop* de propositions atomiques et soit Φ une proposition logique décrivant un invariant $P(\Phi)$ sur le même ensemble de propositions atomiques *Prop*. L'algorithme 1 ci-après permet de déterminer si ST vérifie $P(\Phi)$ ou non.

L'objet de ce devoir consiste à implémenter l'algorithme 1 de vérification d'invariant, en respectant les consignes suivantes :

1. le langage d'implémentation sera `C++` ou `python` ;
2. les systèmes de transition seront implémentés sous la forme d'une classe d'objets ou sous la forme d'une structure ;
3. le choix de l'implémentation d'une proposition logique est libre ;
4. vous devez prévoir d'implémenter une fonction pour tester la relation de satisfaction $s \models \Phi$, pour chaque état $s \in S$;
5. le programme sera testé sur l'exemple d'un système de transition modélisant deux processus concurrents, dont l'entrée dans le processus critique est contrôlée par un sémaphore binaire, avec la propriété d'exclusion mutuelle, ainsi que sur un deuxième exemple de votre choix ;
6. la complexité temporelle du programme devra être optimale ;
7. le travail sera réalisé en binôme ou en monôme, mais pas en trinôme ;
8. les fichiers sources seront présentés et commentés avec soin ;
9. vous rédigerez de plus un court rapport pour expliquer vos choix, notamment concernant l'implémentation des systèmes de transition et des propositions logiques ; ce rapport aura la forme d'un document pdf ;
10. le travail sera rendu sous la forme d'une archive contenant les sources nécessaires à une bonne exécution du programme, sur la plateforme MADOC, avant le 7 novembre 2023 (23h59).

Algorithme 1 Vérification d'invariant

Entrée : système de transition fini ST et proposition logique Φ.
Sortie : « OUI » si ST satisfait toujours Φ, autrement « NON » et un contre-exemple.

Ensemble d'états $R \leftarrow \varnothing$ // l'ensemble des états accessibles
Pile d'états $U \leftarrow \varepsilon$ // ε désigne la pile vide
Booléen $b \leftarrow$ VRAI // tous les états de R vérifient Φ
Tant que $I \setminus R \neq \varnothing \wedge b$ **faire**
  choisir $s \in I \setminus R$ // on choisit arbitrairement un état initial qui n'est pas dans R
  visiter(s) // on appelle la procédure de balayage
Fin Tant que
Si b **alors**
  renvoyer « OUI » // ST satisfait toujours Φ
Sinon
  renvoyer (« NON », U) // la pile U fournit un contre-exemple
Fin Si

Procédure visiter(état s)
  $push(s, U)$ // on pose s sur la pile
  $R \leftarrow R \cup \{s\}$ // on marque s comme accessible
  Répéter
    $s' \leftarrow top(U)$ // s' est le premier élément de la pile
    Si $Post(s') \subseteq R$ **alors**
      $pop(U)$ // on retire le premier élément de la pile
      $b \leftarrow b \wedge (s' \models \Phi)$ // on vérifie la validité de Φ en s'
    Sinon
      choisir $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R \leftarrow R \cup \{s''\}$ // s'' est un nouvel état accessible
    Fin Si
  Jusqu'à $(U = \varepsilon) \vee \neg b$
Fin Procédure