Nantes Université 7 novembre 2023

Modélisation et Vérification des Systèmes Concurrents

Devoir Maison n°1

Lien du github :
https://github.com/Stevehunn/Invariant-verification

Nicolas Bitaillou

M2 ALMA

Introduction:

Le projet est codé en Python et à pour but de vérifier un invariant dans un système de transition. Pour cela l'algorithme en pseudo code comprendra la méthode algo1() et la méthode visiter(). Le programme devra prendre en entrée un système de transition **ST** et une proposition logique **phi.** Le programme devra retourner "Oui" si le système de transition satisfait la proposition logique et "Non" s'il n'est pas satisfait, suivis d'un contre-exemple.

Présentation du code

Tout d'abord nous devons définir les états disponibles ainsi que leur liste de transition. Pour ce faire, les états seront représentés par un nom et un label. Le nom servira d'identifiant et le label contiendra les informations contenues dans l'état.

Exemple: State1 = State(1,{"n1","n2"})

Afin de gérer les différentes conditions et possible changement de valeur leur d'une transition entre deux états, l'utilisation d'une classe Condition fut nécessaire. Elle permet de vérifier si l'on peut aller d'un état a vers un état b et faire par la suite une modification de variable si c'est nécessaire quand les conditions sont satisfaites.

Exemple : condition y>0

La gestion des données dans la pile ainsi que dans R pour les états déjà visités seront mis dans une classe Stack de type liste. La classe comprend diverses méthodes permettant de gérer l'ajout ou la suppression de données, en fonction des besoins de la pile et de R.

Proposition Logique

Les propositions logiques Not, And, Or et Unary ont été implémentées en classes distinctes mais avec comme héritage la classe property pour faciliter son implémentation. Les classes disposent de la méthode eval() qui prend en entrée une ou deux propositions selon la proposition logique utilisée. Pour les propositions logiques And et Or il est nécessaire de fournir deux propositions et cela retournera l'évaluation des propositions. Le projet utilisera donc la proposition phi suivante :

Or(Not(Unary({"c1"})),Not(Unary({"c2"})))

Vérification d'invariant

L’algorithme fourni dans l’énoncé est présent dans le dans les méthodes verification_invariant() et visiter(). Durant leurs exécutions, il est nécessaire de faire appelle aux méthodes post() pour vérifier si l’état est dans R et la méthode phi() qui doit vérifier si le contenu de satisfait la proposition phi.

Réalisation du Programme

Afin de réaliser le projet plusieurs essais ont été nécessaire pour obtenir un programme fonctionnelle et généraliste. La première idée fut de réaliser le code avec des transition en « dur » ce qui rendrait les futurs implémentations difficile. Par la suite, mettre en place une liste de liste afin d’avoir un identifiant dans la première liste et les transitions pour cet état dans la seconde liste. Cela peut néanmoins impacter les performances lors du parcours des transitions et lors des vérifications.

Conclusion

La modélisation des transition systèmes a posé pas mal de problème occasionnant de nombreuses tentatives pour obtenir un résultat satisfaisant, les précédentes tentatives étant le Main2 et Main3. Le code est commenté de manière à expliquer le contenu des classes et leur utilité de façon succincte. Les autres commentaires présents durant la lecture du code sont là pour expliquer les subtilités et la raison d’une telle implémentation.