Rapport

Modélisation et vérification des systèmes concurrents (X3IA110)

GERARD Kylian

Nantes Université

Sommaire

Introduction

La vérification d'invariants joue un rôle essentiel dans le domaine de l'informatique, en particulier dans le contexte de la conception et de la validation des systèmes de transition. Un invariant est une propriété logique qui doit être maintenue tout au long de l'exécution d'un système.
Ce rapport exposera notre approche pour implémenter l'algorithme de vérification d'invariant en respectant les consignes fournies, notamment le choix de Python comme langage de programmation, la représentation des systèmes de transition, la gestion des propositions logiques, et la mise en place de mécanismes de test de satisfaction de l'invariant. Enfin, nous évaluerons les performances de notre programme en le testant sur deux exemples concrets de systèmes de transition et en expliquant nos choix en matière d'implémentation.

I – Système de transition

Les systèmes de transition ont été modélisé à l'aide d'un ensemble d'états (states), d'un ensemble d'états initiaux (initial_states), et d'une fonction de transition, représenté par des pairs entre un état source, et un ensemble des états d'arrivés associés. (transition_function).
Nous avons décidé de ne pas modéliser l'ensemble L (label), l'ensemble d'action (Act), ainsi que l'ensemble de proposition (Prop), car tous ces ensembles n'interviennent pas dans l'algorithme de vérification d'un invariant.

II – Proposition logique

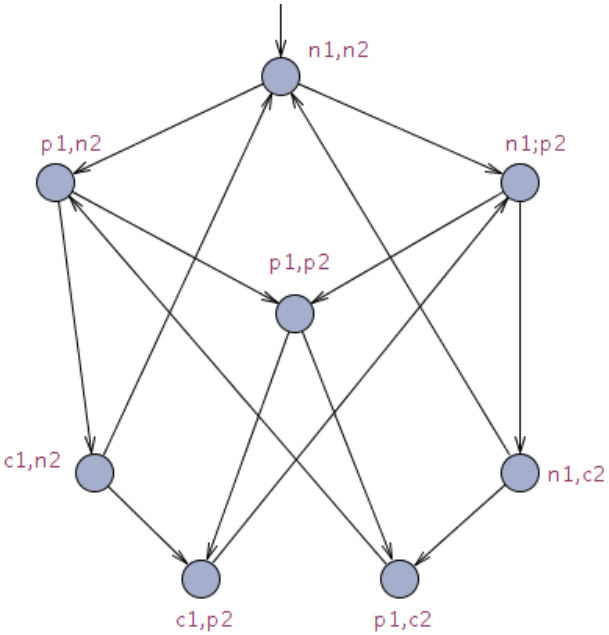
Nous avons choisi d'utiliser une expression lambda prenant un état en entrée, et renvoyant un booléen. Cela permet de définir simplement un prédicat sur un état, qui pourra être vérifier ou non sur les états de notre système de transition.

III – Complexité temporelle

Un algorithme de vérification d'un invariant, nous ayant été fournis, il est compliqué de pouvoir améliorer la complexité de celui, afin de le rendre optimale. En effet, il existe peut-être d'autres algorithmes, utilisant d'autres méthodes, et ayant une meilleure complexité.
Nous avons donc plutôt essayé d'optimiser l'algorithme fournis.
Étant donné que cet algorithme est hautement répétitif, en particulier lors de la vérification de l'invariant pour un état donné qui reste constant pour un même état, nous avons choisi de stocker les résultats obtenus au fur et à mesure de l'exécution de l'algorithme. Cela nous permet d'effectuer les calculs de vérification de l'invariant pour chaque état au maximum une fois.

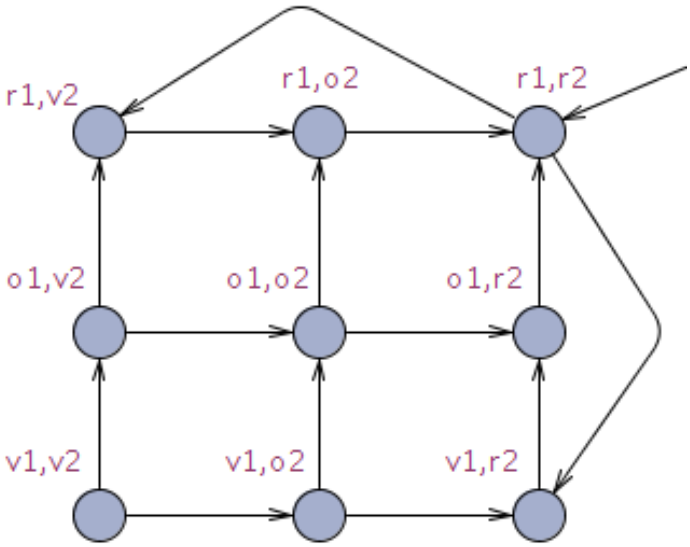
IV – Exemples

- Le premier système de transition modélise deux processus, dont l'entrée dans le processus critique est contrôlée par un sémaphore binaire, avec la propriété d'exclusion mutuelle $\neg(c1 \wedge c2)$.



Ce système de transition valide bien l'invariant.

- Le deuxième système de transition modélise un système de deux feux tricolore (vert, orange, rouge). On cherche ici, à interdire le fait que les deux feux permettent le passage des voitures en même temps $\neg(v1 \wedge v2) \wedge \neg(v1 \wedge o2) \wedge \neg(o1 \wedge v2) \wedge \neg(o1 \wedge o2)$



Ce système de transition valide bien l'invariant.

V – Conclusion

Ce travail nous a permis de mettre en œuvre un algorithme de vérification d'invariant en respectant les directives énoncées tout en présentant des solutions pour gérer les systèmes de transition et les propositions logiques. Bien que l'optimisation de la complexité temporelle ait ses limites, notre implémentation s'est avérée efficace dans les exemples testés, démontrant ainsi la faisabilité de la vérification d'invariants dans un contexte informatique.
Dans l'ensemble, ce travail nous a permis d'acquérir des compétences pratiques dans le domaine de la vérification d'invariants et de la modélisation de systèmes de transition. Ces connaissances seront précieuses pour aborder des systèmes complexes nécessitant des propriétés de sureté.