Vérification et Validation - TP Model checking (M2ISTR)

Camille Coquand

7 Décembre 2023

1 Objectifs du TP

L'objectif de cette séance de TP est de vous familiariser avec l'utilisation d'un *model checker*. Nous utiliserons pour cette séance le model checker **muse** du logiciel TINA développé au LAAS-CNRS[1]. Ce TP comporte deux exercices :

- le premier porte sur l'étude d'un système ferroviaire et a pour objectif d'illustrer les différents types de propriétés énoncés en cours
- le second porte sur l'étude de la fréquentation d'une grotte préhistorique et a pour objectif d'illustrer la sensibilité de la vérification au modèle étudié

À l'issue de la séance un compte-rendu vous est demandé. Sa note sera couplée à celle de vos TPs de *Test logiciel* (elle comptera pour 30% de la note globale). Les attendus du compte-rendu sont les suivants :

- Le format de nom sera le suivant : *nom1_nom2_gpTp_m2istr.pdf*.
- Vous êtes limités à deux pages maximum.
- Vous rédigerez une introduction et une conclusion rapide présentant les objectifs de la séance de TP.
- Vous devrez faire figurer les traductions CTL des exigences des deux exercices, en précisant pour l'exercice 2 le modèle utilisé. Vous devrez également donner vos réponses pour la question clôturant chaque exercice.
- Chaque formule demandée dans le TP est associée à sa numérotation dans le sujet.
- Pour chaque formule indiquer si elle est vraie ou fausse.
- Indiquer le type de propriété associé à chaque formule (une explication concise est attendue).

[1] https://projects.laas.fr/tina/index.php

- Il vous est demandé d'illustrer les deux dernières propriétés de l'exercice 1 par un schéma. Vous inspirer des schémas vus en cours est fortement conseillé. Vous pouvez par exemple représenter des propriétés bien choisies avec des couleurs pour mettre en évidence les états où celles-ci sont vraies.

Des changements de consignes pourraient éventuellement arriver. Dans ce cas, appliquez les consignes qui auront été précisées en cours si celles-ci sont en contradiction avec l'une des consignes présentes dans ce document. Le barème de notation pourra être communiqué à la demande des élèves après notation des compte-rendus.

2 Utilisation de Tina

Pour utiliser Tina, ouvrez un terminal et tapez **nd** puis deux fois rapidement sur **tab** pour obtenir la commande complète. Exécutez la commande du type **nd-***.

Pour ouvrir un fichier, cliquez sur **File -> open** et cherchez le fichier que vous souhaitez charger. Pour générer le graphe des marquages d'un réseau de Petri chargé, cliquez sur **Tools -> state space analysis**. Un menu *tina options* s'ouvre. Choisissez l'option **marking graph (-R)** : plusieurs choix s'offrent à vous. Dans **output** choisir **kts(.ktz)** construit la structure de Krikpe associée au graphe des marquages. Valider puis faire un clic droit sur la nouvelle fenêtre et cliquer sur **model check MMC** vous permet d'ouvrir le model checker **muse** en mode graphique que nous utiliserons dans ce TP (pour vérifier que vous avez ouvert le bon programme, le mot **muse** apparaît dans le nom du buffer). L'option **verbose** vous ouvre une fenêtre textuelle explicitant les différents éléments de définition du graphe (structure, contenu des états,etc.). Enfin, l'option **kts(.aut)** vous donne une description textuelle du graphe et permet de l'afficher dans une nouvelle fenêtre graphique. Le menu **Tools** contient également un **stepper simulator** vous permettant de faire évoluer manuellement le réseau de Petri (pour sortir du mode stepper, cliquez sur **File -> return to editor** ou pressez **Ctrl-q**).

Pour pouvoir utiliser les opérateurs **EF**, **EG**, **AG** et **AF** dans muse, les lignes suivantes doivent être exécutées :

```
prefix AF g = min x | g \/ (<T>T /\ [T]x); ## Inev
prefix EF g = min x | g \/ <T>x;           ## Pot
prefix AG g = max x | g /\ [T]x;           ## All
prefix EG g = max x | g /\ ([T]F \/ <T>x); ## Some
```

Dans **muse**, exécutez la commande suivante :

```
source ctl.mmc;
```

Cette commande permet de charger la définition des opérateurs définis dans le fichier **ctl.mmc** (vous pouvez ouvrir ce fichier en textuel pour voir son contenu, vous pouvez également retrouver son contenu dans la slide 39 de votre cours). Pour plus de détails sur le fonctionnement de muse, tapez **muse -help** dans un terminal ou allez voir la page associée sur le site web de TINA.

Muse propose trois mode différents pour donner le résultat de la vérification d'une propriété : un mode booléen qui vous indique si une propriété est vraie pour la structure de Kripke étudiée (modèle + état initial), un mode cardinal qui vous donne le nombre d'états où la propriété est vraie, et un mode *set* qui vous donne explicitement les états dans lesquels une propriété est vérifiée. Pour activer ces différents modes (exclusifs entre eux), exécutez les commandes suivantes :

```
output bool;
output card;
output set;
```

Il vous est **très fortement conseillé** de taper vos formules dans un fichier txt en parallèle pour vous simplifier la rédaction de certaines formules, l'interface graphique de muse ne permettant pas de copier du texte. Conservez l'ensemble des formules dans un fichier pour pouvoir discuter avec l'encadrant de TP.

3 Échauffement

Pour prendre en main muse, il vous est proposé de lire de faire l'exercice proposé pour les systèmes **pont.ndr** et **piscine.ndr** par François Vernadat dans sa série de TP : échauffement. Les fichiers Tina à exécuter sont disponibles dans le dossier du TP.

4 Exercice 1 - Un problème de trains

Ouvrez le fichier **systeme_trains.ndr**. Ce fichier contient un réseau de Petri modélisant le trajet de deux trains entre 4 villes.

Le fonctionnement du réseau de train est le suivant. Le train 1 peut se déplacer de la Ville0 vers la Ville2. Le train 2 peut se déplacer de la Ville1 vers les villes Ville2 et Ville3. Pour se rendre à la Ville2, les deux trains partagent une portion de trajet commune. Une solution a été proposée pour éviter toute collision entre ces deux trains, en intégrant un système d'aiguillage à l'intérieur du réseau ferroviaire. Afin de vérifier le bon fonctionnement de ce réseau ferroviaire un modèle réseau de Petri a été synthétisé. Nous nous sommes procurés (nous ne préciserons pas comment) un extrait du cahier des charges du projet. Le rédacteur, n'ayant pas suivi de cours de Model checking, a rédigé les spécifications en langage naturel, ce qui vous le savez, rend complexe l'automatisation de la vérificaton de ces spécifications. L'extrait en question est le suivant :

1. Au départ de la Ville1, le train 2 peut arriver dans la Ville2 ou dans la Ville3.
2. Deux trains ne peuvent pas se trouver en même temps sur le tronçon commun.
3. Un train ne peut pas être dans deux villes en même temps.

4. Le train 1 au départ de la Ville2 arrivera toujours dans une autre ville.

5. Le tronçon commun, s'il est occupé, sera toujours libéré.

6. Lorsque les deux trains sont au départ des villes 0 et 1, le train 2 peut arriver à la Ville3 avant que le train 1 soit arrivé à la Ville2.

7. Depuis la Ville0, le train 1 ne peut pas arriver en Ville2 sans passer par le tronçon commun.

8. Il est possible pour le train 1, après un départ depuis la Ville0 vers la Ville2, de ne pas bouger de la Ville2.

9. Arrêté en Ville2, le train 1 réalise nécessairement un aller-retour vers la Ville0.

Dans cet exercice, nous considérons que le modèle du système proposé est fidèle au fonctionnement du système réel. Autrement dit, toute propriété non vérifiée sur le modèle est considérée non vérifiée sur le système réel.

Après avoir analysé la modélisation du système proposé, traduisez ces propriétés en logique temporelle arborescente (CTL) et vérifiez à l'aide du model checker **muse** ces propriétés. Pour la propriété **7**, on utilisera l'opérateur *until* (noté **U**) : **f U g** signifie que **f** est vraie jusqu'à ce que **g** soit vraie.

À la fin de votre analyse, une propriété de *sûreté* n'est pas vérifiée. Proposez une modification du modèle proposé afin que cette propriété soit satisfaite. Implémentez dans TINA votre solution, générez le graphe des marquages et vérifiez la validité de votre proposition.

5 Exercice 2 - Fréquentation d'une grotte préhistorique

La conservation des grottes contenant des peintures préhistoriques est un sujet complexe. Il y a d'une part besoin des recettes causées par les visites de ces monuments, et d'autre part le besoin de limiter la quantité de visiteurs par jour et par heure à l'intérieur de la grotte. Heureusement, les réseaux de Petri sont un outil particulièrement pertinent pour modéliser la fréquentation d'une grotte pour une journée. Nous nous intéressons ici à la grotte Grotte0 contenant 3 salles différentes à visiter. Les limites imposées pour préserver les conditions de température et d'hygrométrie à l'intérieur de Grotte0 sont les suivantes (ces données sont fictives et ne sont pas représentatives des contraintes réelles des grottes préhistoriques) :

1. Le nombre de visiteurs présents en même temps dans Grotte0 doit être inférieur ou égal à 45

2. Le nombre de visiteurs présents dans une salle doit être inférieur ou égal à 20

3. Les visites se font avec un guide avec des groupes de 14 personnes (le guide est modélisé comme un visiteur par la suite)

4. Le nombre de visiteurs par jour est limité à 150

Pour vérifier que les spécifications sont satisfaites, deux modèles différents pour Grotte0 sont proposés (fichiers **grotte0_v1.ndr** et **grotte0_v2.ndr**). Il vous est demandé de modéliser ces propriétés en CTL et de les vérifier, si possible, sur les deux modélisations proposées. Pour modéliser la propriété 4, on pourra s'aider de l'opérateur suivant :

```
op dead = - <T>T; # deadlocks
```

Cet opérateur caractérise des états pour lesquels le réseau de Petri étudié est dans un blocage, c'est à dire que pour un tel état, aucune transition n'est sensibilisée.

Après avoir fait votre vérification, conclure quant à la validité des modèles proposés. Proposez un modèle plus proche de la réalité permettant de satisfaire toutes les exigences présentées. Vous pourrez pour cela vous baser sur l'un des deux modèles roposé et le modifier.