Mini Mémoire
Le Model-Checking de CTL

BUI QUANG PHUONG Linh – 000427796
Promoteur : GEERAERTS Gilles

Université Libre de Bruxelles

Résumé La vérification de modèles, plus communément appelée via son appellation anglaise *Model-Checking*, est un système de vérification automatique qu'un système informatique ou électronique satisfasse une certaine propriété. Celle-ci est généralement utilisé afin de prouver la bonne fonctionnalité du système ou dans le cas contraire, de détecter des bugs ou des dysfonctionnements. Plusieurs types d'algorithmes et de logiques permettent d'effectuer un Model-Checking. Nous allons principalement nous intéresser au Model-Checking de CTL mis en place durant les années 80.

Table des matières

1 Introduction

1.1 Contexte et point de vue global

Les dernières décennies entrainant l'évolution exponentielle de la technologie et de l'informatique, la société actuelle fait régulièrement usage de systèmes automatisés afin d'assurer le bon fonctionnement et la fiabilité des programmes, machines et autres divers systèmes informatiques. Ces systèmes automatisés sont appelés *Model-Checking*.

Le principe de base du model-checking est la vérification qu'un certain système satisfasse une certaine propriété. Le système sera représenté par un modèle M tandis que la propriété fera office d'une formule Φ. Dans un premier temps, la modélisation du comportement dynamique du système est nécessaire avant d'être suivie par la vérification de la formule de la propriété grâce à l'algorithme de model-checking qui déterminera si le modèle satisfait bien la formule.

$$\boxed{M \vDash \Phi}$$

Lorsque le système ne satisfait pas la propriété évaluée, le model-checking a donc repéré un bug ou un dysfonctionnement. En guise d'information, chaque année le coût des bugs vaut environ 64 milliards de dollars rien qu'aux Etats-Unis. La création et la mise en place des model-checking est donc impérative pour réduire et minimiser ces coûts.

Illustrons cela par un exemple plus concret : un distributeur de boissons. Dans ce cas, il est indispensable que le montant demandé pour une certaine boisson soit mise dans la machine afin que le client puisse récupérer la boisson. Il s'agit d'un système de model-checking qui va être utilisé afin de vérifier que cette propriété soit satisfaite. Un autre exemple illustrant l'importance d'un model-checking est la fermeture des barrières sur une voie ferrée lors du passage d'un train. Celle-ci doit être réalisée automatiquement lorsqu'un train est sur le point de traverser une voie ferrée. Dans ce cas, une petite erreur dans le model-checking peut avoir une très grande ampleur concernant la sécurité des passants. Il est donc impératif de vérifier la bonne fonctionnalité du model-checking.

1.2 Avantages & désavantages

Avantages du model-checking Un model-checking possède plusieurs avatanges par rapport à l'utilisation d'éventuelles autres techniques de vérifications et de détection d'erreur. Voici une liste non exhaustive de ces avantages :

- Il s'agit d'une approche générale de vérification applicable dans tous les domaines tels que les systèmes embarqués ou ingénierie logicielle.
- La phase de vérification est automatique. La seule action nécessaire afin de faire fonctionner le model-checking de l'utilisateur est de l'activer. Le programme s'occupe du reste.
- Le model-checking est **efficace**. Contrairement aux simples tests, il va également prouver la bonne fonctionnalité du système et dans le cas contraire, nous renvoyer un contre-exemple d'une exécution du système qui falsifie la propriété et ainsi, détecter le(s) bug(s).
- Dans le cas général, le model-checking est cost-efficient.
- Le model-checking agit sur tout le système et non qu'une seule partie, il s'agit donc d'une méthode exhaustive. Néanmoins, si l'utilisateur le souhaite, il est tout de même possible d'utiliser le model-checking pour une vérification partielle et ainsi l'appliquer à des parties de système.

— Utilisation de la logique temporelle (*cfr. section "Logique temporelle"*) qui permet d'exprimer facilement les diverses propriétés.

Désavantages du model-checking Néanmoins, le model-checking est une solution discutable sur certains points :

— Le model-checking grandit exponentiellement par rapport au nombre de processus actifs (taille du système). Le modèle devient donc très vite surchargé et surpasse la capacité de la mémoire. Ce problème est appelé *State explosion problem*. Par conséquent, le model-checking ne pourrait donc pas croître.
— Le model-checking s'applique généralement sur des systèmes finis et n'est pas adapté aux systèmes non-finis. Hors, la possibilité de tomber sur un système non-fini (ayant donc une infinité de valeurs possibles pour les variables) n'est pas écartée.
— Comme son nom l'indique, le model-checking exécute une vérification du modèle du système et non du système en lui-même. Les résultats obtenus pour le modèle correspondent en grande partie au système réel, néanmoins, il est nécessaire d'utiliser des techniques supplémentaires comme des tests pour trouver les éléments qui pourraient différer entre le modèle et le système réel, que ce soit au niveau du hardware tel que des défauts de fabrication ou bien au niveau software tel que des erreurs de codage.
— Conséquence du dernier point : le résultat n'est donc pas garantit à 100% et pourrait donc contenir des erreurs.

2 Point de vue algorithmique

2.1 Graphes et principes d'états

Présentation du principe de systèmes d'états finis et de graphes en quelques mots + Schéma

2.2 Structure de Kripke

Présentation de la structure de Kripke, définitions des différentes notions.

Application - exemple de modélisation à l'aide de la structure de Kripke Exemple d'une structure de Kripke + Schéma. Explication de l'exemple.

2.3 Arbre d'exécution d'une structure de Kripke

Pourquoi utiliser un arbre ? Expliquer pourquoi on passe d'un graphe à un arbre ? Quel est en l'utilité ?

Application - du graphe à l'arbre Traduction de l'exemple de Kripke donné ci-dessus en arbre.

3 Le model-checking de CTL : généralités et logiques

3.1 Introduction à la logique temporelle

Première approche de la notion de logique temporelle, des différentes sortes de logiques temporelles existantes (linéaire ou arborescente), des propositions de logiques temporelles, des formules associés, ...

LTL Aperçu et premier approche de la logique temporelle linéaire LTL

CTL Aperçu et premier approche de la logique temporelle linéaire CTL

3.2 Avantages d'une structure d'arbre

Présentation des différents avantages de CTL

Brève comparaison avec LTL Comparaison (avantages+désavantages) par rapport à LTL

Exemple d'arbre CTL Exemple d'un arbre CTL + explications

3.3 Syntaxe

Présentation détaillée de la syntaxe utilisée par CTL ainsi que ses différentes propriétés

3.4 Sémantique

Présentation de la sémantique de CTL - des relations de satisfactions, des interprétations des différentes formules, etc..

4 Le model-checking CTL : algorithme et implémentation

4.1 Pseudo-code de l'algorithme

Pseudo-code (ou code personnel) simple d'un algorithme de Model Checking CTL.

4.2 Analyse du code

Analyse détaillée du code de l'algorithme et exemple d'application

4.3 Complexité

Complexité du code donné ainsi que d'un model checking de CTL en général