Mini-mémoire
Model Checking CTL

[Cours : INFO-F308]

Paquet Michael - 000410753
Promoteur : Geeraerts Gilles

---

08 Novembre 2017

Table des matières

1 Introduction

Avec l'évolution technologique faisant acte de présence jours après jours, la plupart des services, informatique ou non, sont maintenant gérés via des système informatiques vérifiant leur bon fonctionnement. Mais pour certains services, il est impératif que ces systèmes fonctionnent correctement.

Prenons l'exemple d'une voie ferrée :

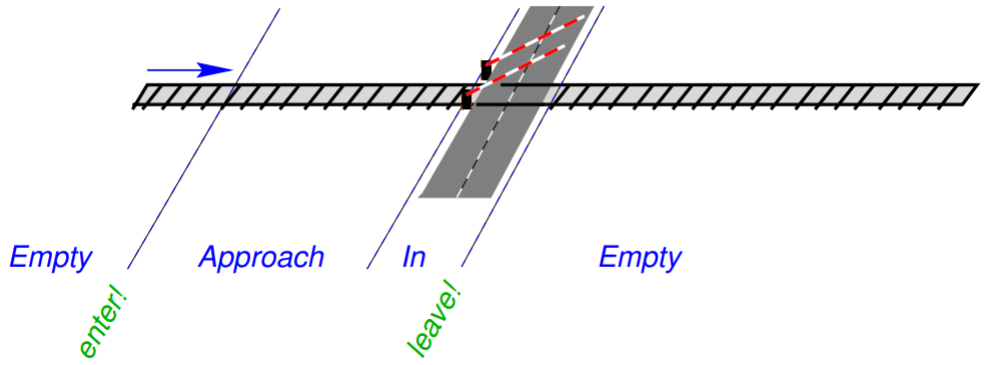

FIGURE 1 – Exemple de modèle

Dans ce cas de figure-ci, il est impératif que les portes soient fermées lorsque le train traverse le passage à niveau. Comme dit précédemment, un système informatique s'occupera de gérer cela.
Mais puisqu'on ne peut pas se permettre la moindre erreur, nous nous devons de vérifier que le système n'échouera jamais, et c'est dans ce cas de figure que le *Model Checking* fut créé et utilisé.

Au cours de cette dissertation, nous passerons donc en revue chacun des points de vue qu'on peut adapter lorsqu'on parle de *Model Checking CTL*.
Le premier point que nous aborderons sera l'aspect analytique du modèle (Qu'est-ce qu'on doit vérifier ?). Lors de ce chapitre, nous allons montrer la façon dont on découpe un système pour en faire des structures propice à l'analyse et à la vérification. Des structures comme les *Kripke models* ou encore les arbre seront bien évidemment abordées. Une fois ces objets observés, nous étudierons la façon dont on peut vérifier l'efficacité d'un système.

Après avoir déterminer la façon dont on peut vérifier l'exactitude d'un système, nous évoquerons la consistance de telles méthodes. Nous prouverons donc de manière rigoureuse l'efficacité du *Model Checking CTL*.

Enfin, nous aborderons la question algorithmique : Comment implémenter un tel modèle ?

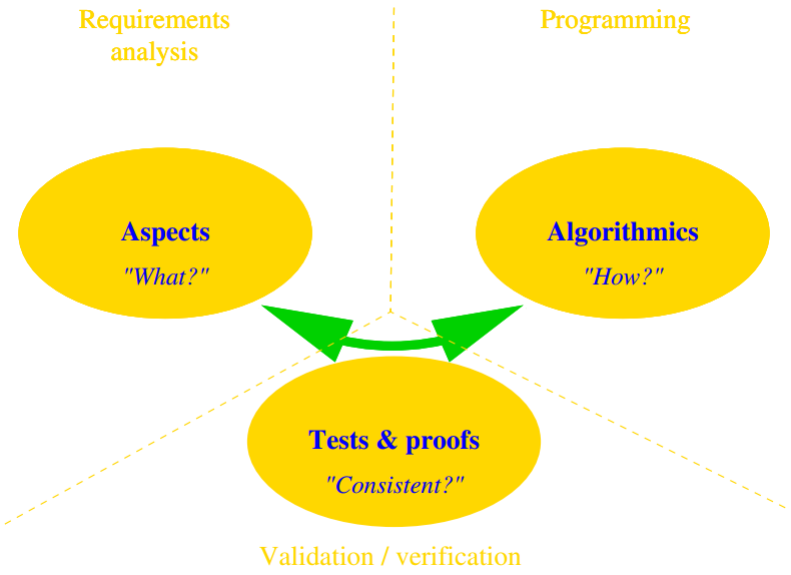

FIGURE 2 – Points de vue

Il est évident que notre sujet ne s'arrêtera pas à ces points-ci. A titre d'exemple, nous comparerons également notre *Model Checking CTL (Computational Tree Logic)* avec le *LTL (Linear temporal logic)*.

2 Chapitre 1

2.1 Du graphe aux arbres de recherche

Pour pouvoir analyser un système et pouvoir vérifier son efficacité, il est utile de le dériver en un graphe. Afin d'imager nos propos, reprenons l'exemple de la voie ferrée :
imaginons donc une voie ferrée sur laquelle aucun train ne circule. La voie est donc vide. On peut constate là un premier état de notre voie ferrée, l'état **vide**. Lorsque notre voie est vide, il se peut qu'à tout moment, un train soit **en approche**, et qu'on doive commencer à fermer les portes. Après quoi la voie sera **occupée** par le train pour qu'ensuite celui-ci reparte et laisse la voie **vide**.

Un tel système peut être représenté sous cette forme :

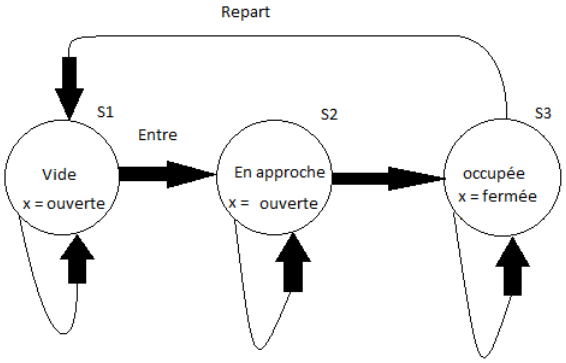

FIGURE 3 – Modèle Kripke

Avec x étant une variable propositionnelle pour déterminer l'état des portes.
Un tel modèle est appelé *Kripke Model*.
Par convention, le modèle *Kripke* :

$$K = (V, S, S_0, I, R)$$

est le modèle *Kripke* dont :

1. **V**{ouverte, se ferme, fermée} est un ensemble fini de propositions atomiques.
2. **S**{vide, en approche, occupée} est en ensemble fini d'état.
3. $\mathbf{S_0}$ est l'état initial.

4. **I : S** $\rightarrow$ $\mathbf{2^V}$ est la fonction qui lie chaque états avec les propositions qui y sont liées.
5. **R** est l'ensemble des relations entre chacun des états.

Nous avons donc ici un graphe infini puisque celui-ci est cyclique. lorsqu'on transformera ce graphe en un arbre de recherche, nous aurons donc également un arbre infini. Faisons le travail de transformer ce graphe en un arbre :

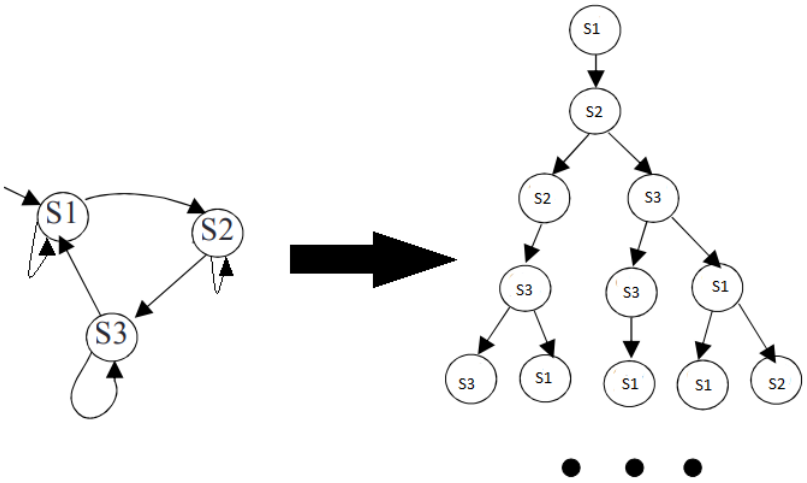

FIGURE 4 – du Modèle Kripke à l'arbre de recherche

Les "..." symbolisent le fait le l'arbre est un arbre infini.

C'est maintenant grâce à de telles structures que nous allons pouvoir analyser notre système et vérifier que celui-ci soit infaillible.

2.2 Computational Tree Logic (CTL)

La syntaxe de la *Computational Tree Logic* utilise des formule booléennes : *True* et *False* sont donc des formules CTL.
Les variables propositionnelles sont des formules CTL.
Ainsi, si φ et ψ sont des formules CTL alors : $\neg\varphi$, $\varphi \wedge \psi$ et $\varphi \vee \psi$ le sont aussi.

En plus de cela, plusieurs opérations sont également des formules CTL :
- EX φ : φ apparait dans un des états suivants.
- EF φ : Dans au moins un chemin, φ apparait dans un état future.

— EG φ : Dans au moins un chemin, φ apparait dans tous les états futures.
— E$[\varphi \cup \psi]$: Dans au moins un chemin, φ apparait jusqu'à ce que ψ apparaisse.

Les opérations ici citées sont à chaque fois des "*There exists*" (EX = Exists next), mais il existe également les opérations "*ForAll*".
Ainsi, AX φ : φ apparait dans **tous** les états suivants. AF, AG, A$\cup$ sont donc aussi des opérations valables.

Voici à titre d'exemple un schéma pour se donner une idée de ce que EF et AF signifient :

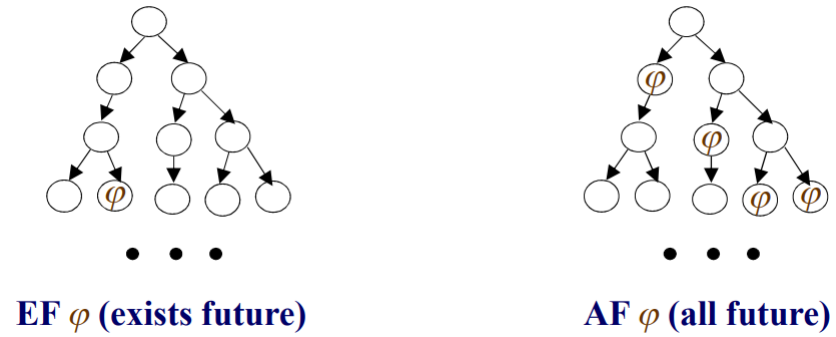

FIGURE 5 – Exemple

A titre d'exemple toujours, essayons d'appliquer ces formules avec l'exemple de la figure 3.
Propriétés validées :
— (AX ouverte)(S_1) : En effet, lorsqu'on démarre de S_1, on peut aller vers S_1 soit S_2, et la variable propositionnelle pour ces deux états est "ouverte".
— (EF ouverte)(S_3)
— (AF fermée)(S_1)

Propriétés non validées :
— (AX fermée)(S_3)

De ces opérateurs découlent pas mal de propriétés. En voici une liste non exhaustive :

- $\neg$ AX φ = EX $\neg\varphi$
- $\neg$ AF φ = EG $\neg\varphi$
- AF φ = A[true $\cup$ φ]
- AG φ = $\varphi\wedge$ AX AG φ
- AF φ = $\varphi\vee$ AX AF φ
- A[false $\cup$ φ] = E[false $\cup$ φ] = φ
- A[φ $\cup$ ψ] = $\psi \vee (\varphi \wedge$ AX A[φ $\cup$ ψ])
- E[φ $\cup$ ψ] = $\psi \vee (\varphi \wedge$ EX E[φ $\cup$ ψ])
- A[φ W ψ] = $\neg$ E[$\neg\psi \cup (\neg\varphi \wedge \neg\psi)$]

Maintenant que nous avons fait un tour non exhaustif de ce qu'était la syntaxe de CTL, abordons la sémantique :

- Soit φ une formule CTL, alors :

  $$K, s \vDash \varphi$$

  signifie que φ est vérifié à l'état **s**. La plupart du temps, K est omis puisque nous toujours au sein du même modèle *Kripke*.

- $\pi = \pi^0\pi^1...$ est un chemin.
- π^0 est l'état initial (la racine).
- π^{i+1} est un des états succédant π^i.

A nouveau, une série de propriétés découle des définitions précédentes :

- AX $\varphi = \forall\pi \cdot \pi^1 \vDash \varphi$
- AX $\varphi = \exists\pi \cdot \pi^1 \vDash \varphi$
- AG $\varphi = \forall\pi \cdot \forall i \cdot \pi^i \vDash \varphi$
- EG $\varphi = \exists\pi \cdot \forall i \cdot \pi^i \vDash \varphi$
- AF $\varphi = \forall\pi \cdot \exists i \cdot \pi^i \vDash \varphi$
- EF $\varphi = \exists\pi \cdot \exists i \cdot \pi^i \vDash \varphi$
- A[φ $\cup$ ψ] = $\forall\pi \cdot \exists i \cdot \pi^i \vDash \psi \wedge \forall j \cdot 0 \leq j < i \Rightarrow \pi^j \vDash j$
- E[φ $\cup$ ψ] = $\exists\pi \cdot \exists i \cdot \pi^i \vDash \psi \wedge \forall j \cdot 0 \leq j < i \Rightarrow \pi^j \vDash j$

Définition : Une formule CTL est $ACTL$ si elle n'utilise que les connecteurs universels temporels (AX, AF, AG, AU).
Définition : Une formule CTL est $ECTL$ si elle n'utilise que les connecteurs existentiels temporels (EX, EF, EG, EU).
Définition : Les formules CTL qui ne sont ni $ACTL$ ni $ECTL$ sont appelées mixtes.

Grâce à la syntaxe et à la sémantique que nous avons évoqué, il est maintenant possible de parler de la sécurité et la vivacité d'un système.

Définition : Un système sûre implique qu'il ne se passera jamais quelque chose de mauvais. AG $\neg$ *bad*
Définition : Un système vivace implique que quelque chose de bien arrivera toujours. AG AF *good*

2.3 CTL Model Checking

2.4 CTL et LTL

3 Chapitre 2