

Etude d'un algorithme de vérification de modèle en Logique Linéaire Temporelle

Dans un monde dont la dépendance à divers technologies est très importante, il est important de s'assurer du bon fonctionnement de ces technologies. C'est ainsi que la vérification de modèle rentre en jeu pour répondre à ces problématiques. Ce sont des techniques de vérifications qui explorent tout les états possibles d'un système de manière force brute. Nous allons donc étudier une famille d'algorithme permettant de résoudre de tel problème.

1 Préliminaires

1.1 Logique Linéaire Temporelle

Definition 1 (Formule de la LTL): On définit les opérateurs suivants:

- $X\phi$: ϕ doit être satisfaite dans l'état suivant (neXt)
- $\psi U \phi$: ψ doit être satisfaite dans tous les états jusqu'à un état où ϕ est satisfait (Until)
- $F\phi$: ϕ doit être satisfaite une fois plus tard (Finally)
- $G\phi$: ψ doit être satisfaite dans tous les états futurs (Globally)

On définit F l'ensemble des formules de la LTL inductivement par:

- $p \in AP \implies p \in F$
- si ψ et ϕ sont des formules de LTL alors $\neg\psi, \phi \vee \psi, \mathbf{X}\psi, \phi U \psi$ sont des formules de LTL

AP un ensemble fini de variables propositionnelles.

Soit $v : \omega \times F \rightarrow \{T, F\}$ une fonction de valuation.

Definition 2 (Monde): On définit un tel objet comme $\omega := w_0, w_1, \dots$ une suite infinie d'état. On écrira $\omega^i := w_i, w_{i+1}, \dots$ un suffixe de ω . On définit $\omega \models f$ via:

- $\omega \models a \Leftrightarrow v(w_0, a) = T$ si a atomique
- $\omega \models \neg f$ si $\neg(\omega \models f)$
- $\omega \models f \vee g$ si $\omega \models f$ ou $\omega \models g$
- $\omega \models \mathbf{X}f$ si $\omega^1 \models f$
- $\omega \models f U g$ si $\omega \models g$ ou $\omega \models f \wedge \mathbf{X}(f U g)$
- $\omega \models \mathbf{F}f$ si $\exists i, \omega^i \models f$
- $\omega \models \mathbf{G}f$ si $\forall i, \omega^i \models f$

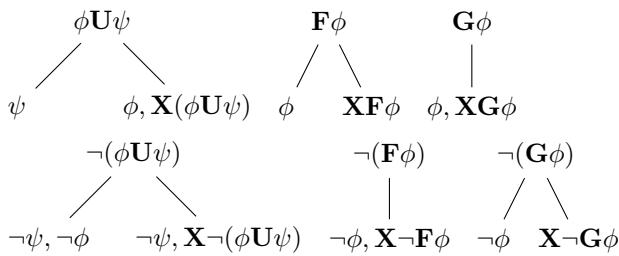
1.2 Méthode des tableaux

Definition 3 (Méthode des tableaux): Algorithme pour prouver qu'une assertion ϕ ayant pour hypothèse (H_n) soit satisfiable. En bref:

- On place ϕ et ses hypothèses dans la racine.
- On applique des règles (R_x) à chaque formule en bout d'arbre qui sont developpables
- Si on trouve des contradictions (des *cycles*) dans toutes les branches de l'arbre (branches fermées), l'arbre est fermé donc la formule est insatisfiable.

$$\begin{array}{ccccc}
 \phi \vee \psi & \neg(\phi \wedge \psi) & \neg\neg\phi & \phi \wedge \psi & \neg(\phi \vee \psi) \\
 / \quad \backslash & / \quad \backslash & | & | & \\
 \phi & \psi & \neg\phi & \neg\psi & \phi, \psi \\
 & & & & \neg\phi, \neg\psi
 \end{array}$$

On définit deux nouvelles règles pour la méthode des tableaux en LTL:



En plus on rajoute des règles supplémentaires: On appelle **X-eventualité** les formules de la forme $\mathbf{X}(aUb)$ et $\mathbf{XF}b$. On notera $x \leq y$ si y est descendant de x (resp $<$ pour descendant strict)

- **LOOP:** Si un noeud v a un ancêtre strict u tq $\Gamma_v \subseteq \Gamma_u$ et pour chaque $\mathbf{X}(aUb)$ ou $\mathbf{XF}b$ dans Γ_u , il existe un noeud w tq $b \in \Gamma_w$ et $u < w \leq v$ alors la formule de départ est satisfiable.
- **PRUNE:** Si $u < v < w$ et ont les mêmes hypothèses Γ et que pour chaque $\mathbf{X}(aUb)$ ou $\mathbf{XF}b$ dans Γ , si on a x tq $b \in \Gamma_x$ et $v < x \leq w$ alors $\exists y, b \in \Gamma_y$ et $u < y \leq v$ et w ferme sa branche.
- **PRUNE₀:** Si $u < v$ et ont les mêmes hypothèses Γ et que il y a au moins une formule X-éventuelle, mais pas de formule de la forme $\mathbf{X}(aUb)$ ou $\mathbf{XF}b$ dans Γ où $\exists x, b \in \Gamma_x$ et $u < x \leq v$ alors v ferme sa branche.
- **TRANSITION:** Si aucune des autres règles ne s'applique, alors un noeud étiqueté par Γ peut avoir un enfant dont l'étiquette est $\Delta = \{a | Xa \in \Gamma\} \cup \{\neg a | \neg Xa \in \Gamma\}$

Proposition 1 (Correction et Complétude): La méthode des tableaux en logique linéaire temporelle est correct et complet

Preuve 1: Par récurrence immédiate (non)

1.3 Automate de Büchi

Definition 4 (Automate de Büchi (BA)): Un automate de Büchi est un 5-uplets (S, I, T, F, Σ) tel que

- S est un ensemble fini d'état
- $I \subseteq S$ un ensemble d'état initiaux
- $F \subseteq S$ un ensemble d'état finaux
- Σ Un ensemble fini de symboles appelé alphabet
- $T : \{S \times \Sigma\} \mapsto \mathcal{P}(S)$ Une fonction de transition.

Cet automate particulier accepte des séquences infinis (donc des mots infinis) ssi le mot passe par un état acceptant une infinité de fois.

Proposition 2 (Test BA Vide): Un BA est non vide ssi il existe un état final qui est atteignable depuis un état initial et appartient à un cycle.

Algorithme: On considère notre automate comme un graphe orienté, qu'on décompose en CFC, puis on fait un DFS depuis les états initiaux et on trouve une CFC non trivial tq il contient un état final et est atteignable par un état initial.

Preuve 2:

Definition 5 (Automate de Büchi généralisé (GBA)): C'est un automate de Büchi où on change l'ensemble des états finaux F par $F_g \subseteq \mathcal{P}(S)$ appelé **condition d'acceptation**.

Un GBA accepte un mot ssi il est passé une infinité de fois par un état dans chaque ensemble d'état acceptant.

Proposition 3 (GBA vers BA): Un ensemble multiple d'état acceptant peut être traduit en un ensemble en construisant un automate par une "counting construction". C'est à dire si $A = (S, I, \{F_1, \dots, F_n\}, \Sigma, T)$, alors il est équivalent à $A' = (S', I', F, \Sigma, T')$ telle que

- $Q' = Q \times \{1, \dots, n\}$
- $I' = I \times \{1\}$
- $F' = F_1 \times \{1\}$
- $\Delta' = \{((q, i), a, (q', j)) | (q, a, q') \in \Delta \text{ et si } q \in F_i, j = i + 1 \pmod{n} \text{ sinon } j = i\}$

qui est bien un automate de Büchi (non généralisé) qui accepte les mêmes mots.

Preuve 3:

Proposition 4 (Intersection de BA): On définit l'automate reconnaissant l'intersection de deux langages comme $A' = (S', I', F', \Sigma, T')$ telle que

- $S' = S_1 \times S_2 \times \{1, 2\}$
- $T' = T'_1 \cup T'_2$
 - $T_1 = \{((q_1, q_2, 1), a, (q'_1, q'_2, i)), (q_1, a, q'_1) \in T_1, (q_2, a, q'_2) \in T_2, q_1 \in F_1 \Leftrightarrow i = 2, i \in [|1, 2|]\}$
 - $T_2 = \{((q_1, q_2, 2), a, (q'_1, q'_2, i)), (q_1, a, q'_1) \in T_1, (q_2, a, q'_2) \in T_2, q_2 \in F_2 \Leftrightarrow i = 1, i \in [|1, 2|]\}$
- $I' = I_1 \times I_2 \times \{1\}$
- $F' = \{(q_1, q_2, 2), q_2 \in F_2\}$

Il reconnaît l'intersection des langages reconnus par les automates de Büchi

Preuve 4:

2 Résolution

2.1 Présentation

Pour résumé le déroulé de l'algorithme, nous devons faire en supposant l'implémentation de la méthode des tableaux en LTL faites et des automates de Büchi.

1. Création d'un automate A_m correspondant au cahier des charges d'une technologies, sous forme des différents états dans lequel la technologie sera.
2. Création d'un automate $A_{\neg\varphi}$ où φ représente les différentes contraintes que notre cahier des charges doit respecter, sous forme d'une formule logique en logique linéaire temporelle.
3. Création de l'automate Syncronisé $A_m \times A_{\neg\varphi}$
4. Si l'automate résultant est vide, alors c'est bon, sinon on renvoie un contre exemple.

Remarques:

- Nous supposerons l'étape 1 donné, l'étape 3 et 4 est immédiat par ce qu'on a défini en préliminaire.
- Intuitivement, Le langage des A_m représente toutes les exécutions de M et celui de $A_{\neg\varphi}$ représente les exécutions ne satisfaisant pas la formule φ . Le langage du produit est donc vide ssi tous les chemins de M satisfont φ . Sinon il existe un mot en commun, qui nous donne l'erreur.
- Pour l'étape 1, en théorie l'automate est donné après transformation d'un système de transition \mathcal{M} en un automate A_M

2.2 Creation de $A_{\neg\varphi}$

On présente ici l'algorithme de Gerth