Vérification parallélisée de propriétés temporelles sur des traces d'exécution, par analyse dynamique formelle

Antoine Ferlin, Philippe Bon, Virginie Wiels, Simon Collart-Dutilleul

FERLIN, Antoine, BON, Philippe, WIELS, Virginie, COLLART-DUTILLEUL, Simon, 2015, Vérification parallélisée de propriétés temporelles sur des traces d'exécution, par analyse dynamique formelle, Approches Formelles dans l'Assistance au Développement Logiciel, Bordeaux, France, 2015-06-09, 15p

Vérification parallélisée de propriétés temporelles sur des traces d'exécution, par analyse dynamique formelle

Antoine Ferlin* antoine.ferlin@ifsttar.fr
Philippe Bon* philippe.bon@ifsttar.fr
Virginie Wiels† virginie.wiels@onera.fr
Simon Collart-Dutilleul* simon.collart-dutilleul@ifsttar.fr

Résumé

Les méthodes de vérification peuvent être classées suivant deux critères : une méthode peut être statique ou dynamique, ainsi que formelle ou informelle. Ce papier poursuit des travaux de thèse sur la vérification de propriétés temporelles sur des traces d'exécution par analyse dynamique formelle. L'approche proposée consiste à transformer une propriété LTL en automate de Büchi et à exécuter ce dernier sur une trace pour l'analyser. Le problème de fin de trace lié à l'utilisation de LTL sur des traces finies peut être contourné par le calcul d'informations statistiques à condition que la propriété suive un patron prédéfini. Pour des traces de très grande taille, cette approche est bien adaptée, mais nécessite que la trace soit vérifiée séquentiellement. Cet article propose de remédier à ce problème, en découpant la trace en plusieurs sous-traces analysables séparément, suivant une stratégie définie, ce qui permet un gain de temps significatif.

1 Introduction

Le développement de logiciels critiques est contraint par des standards de certification dépendant des domaines d'applications. Par exemple, le standard pour les logiciels avioniques est le DO-178 ; dans le ferroviaire, on utilise la norme IEC 50128. Bien que les objectifs pour chaque étape du développement soient définis par les standards, il incombe aux entreprises de choisir les méthodes de vérification permettant d'atteindre ces objectifs. Les standards peuvent proposer à titre indicatif des méthodes connues.

Une façon de classer les techniques de vérification est de considérer deux critères : statique ou dynamique, formelle ou non formelle. Une méthode statique signifie que la vérification s'effectue sans l'exécution du programme. À l'inverse, une méthode dynamique nécessite l'exécution du programme. Une méthode formelle mettra en œuvre une analyse formelle en s'appuyant sur un langage possédant une sémantique formelle. À titre d'exemples, les revues de code sont statiques et non formelles, le test est classiquement dynamique et non formel. Les méthodes formelles sont classiquement statiques : B [1], model checking [8], interprétation abstraite [9]... Nous nous intéressons dans cet article aux méthodes formelles dynamiques [1].

Ces travaux font suite à une thèse Cifre faite à Airbus et à l'Onéra, à propos de la vérification de propriétés temporelles [13, 12]. Pendant ces travaux, plusieurs logiciels embarqués ont été analysés pour extraire les propriétés difficiles à vérifier par les techniques de vérification existantes. Un langage adapté au contexte industriel a été défini pour formaliser ces propriétés. Ce langage combine la logique temporelle linéaire (LTL) et des expressions régulières, particulièrement adaptées aux propriétés de séquence. La thèse a permis de définir une méthode de vérification

*IFSTTAR/COSYS-ESTAS, 20 Rue Élisée Reclus, BP 70317, 59666 Villeneuve d'Ascq Cedex
†ONERA/DTIM, 2 avenue Édouard Belin, BP74025,31055 Toulouse

1. Conférences *Runtime Verification*, 2001-2015, `www.runtime-verification.org`

outillée adaptée à ces propriétés. L'approche développée fait partie des méthodes de *Runtime Verification*. La méthode développée se compose de deux étapes : la transformation de la propriété formalisée en automate de Büchi non-déterministe à l'aide de *Ltl2ba* [14], puis l'exécution de cet automate sur une trace d'exécution du programme à analyser. Comme LTL a une sémantique sur des traces infinies, une trace d'exécution finie est rendue infinie par bouclage sur le dernier état de la trace. Les effets de bord sont contrés par le calcul d'informations statistiques sur la trace pour guider l'interprétation du résultat, pour les cas litigieux. Le lecteur intéressé par plus de détails sur l'approche, notamment sur les effets de bord et les cas litigieux, pourra lire [13].

Cette approche est envisagée dans le domaine ferroviaire. Nous projetons de l'utiliser pour analyser des traces issues de la plate forme de simulation [2]. Cependant, la plateforme étant en pleine évolution dans le cadre du projet PERFECT [3], nous effectuerons ici des expérimentations sur des traces générées aléatoirement. Pour réduire davantage le temps de vérification lié à cette méthode, ce papier propose une version parallélisée de cette approche.

La section 2 positionnera cette approche suivant l'état de l'art et le contexte industriel. La section 3 résumera l'approche définie en [13]. La section 4 définira quelques notations utiles à la compréhension de ce papier, la section 5 formalisera précisément l'approche classique d'exécution d'un automate de Büchi sur une trace. On s'appuiera sur cette dernière pour formaliser l'approche parallélisée en section 6. Ensuite, la section 7 présentera les expérimentations menées, en mettant en relief les deux approches. Finalement, la conclusion (section 8) de cet article portera sur l'efficacité de l'approche parallèle et sur ses limitations.

2 Contexte

2.1 État de l'art

On peut classer les travaux de *Runtime Verification* en deux catégories : les méthodes *online*, qui effectuent des vérifications pendant l'exécution du programme, et les méthodes *offline* qui effectuent des vérifications sur des traces d'exécution de programme, à savoir des séquences d'états de programme. Il est à noter qu'un état correspond à un instant associé à l'ensemble des variables du programme et de leur valeur.

De nombreux travaux concernent la vérification *online*, en particulier pour les programmes Java [19, 11, 18, 17] et les communautés de programmation orientée aspects [23]. Certains travaux sont basés sur des règles de réécriture des propriétés [17] ou des techniques spécifiques de transformation de propriétés LTL en machines à état [16, 10].

La réduction du temps de vérification est un enjeu essentiel pour pouvoir traiter des logiciels industriels. De nombreux efforts sont faits en verification *online* dans ce sens. [5] propose de réduire le nombre de moniteurs à l'aide de l'analyse statique. [4] définit une approche parallélisant la vérification de plusieurs propriétés LTL trivaluées en utilisant le GPU [4] pour encoder les moniteurs. [25, 24, 20] dissocient les moniteurs des programmes à analyser.

L'approche proposée dans ce papier est une approche *offline*, les traces d'exécution seront issues à terme d'un simulateur ferroviaire. Le choix d'une approche de vérification a posteriori a été fait pour plusieurs raisons. L'une des plus importantes est qu'il ne faut pas perturber la simulation en ajoutant des processus dédiés à la vérification. En effet, nous souhaitons limiter le ralentissement de l'exécution dans l'application temps réel pour éviter des conséquences sur la véracité d'une propriété. Dans ce contexte de vérification a posteriori, trois facteurs jouent sur le temps de vérification : la taille de la trace, le nombre de variables dans la trace, et le format de la trace (binaire, Xml, ...). Ce dernier facteur n'est cependant pas optimisable puisque le simulateur est propriétaire, ce qui interdit également l'application d'une approche GPU qui nécessite une modification de la plateforme.

2. Plateforme propriétaire Ersa, European Rail Software Applications, `http://www.ersa-france.com`
3. `http://perfect.ifsttar.fr/Site`
4. *Graphic Processing Unit*

Dans plusieurs travaux, les traces sont obtenues par écoute de l'ensemble des variables du programme à analyser, même si seules quelques variables sont utiles pour vérifier une propriété donnée [21, 3]. Cette approche aboutit à la génération de traces imposantes. Il faut rappeler que le temps d'analyse est linéaire avec la taille de la trace et le nombre de variables.

Pour limiter la taille de ces traces, l'analyse statique [12] offre la possibilité de détecter tous les points de programme où un opérande de la propriété à vérifier varie. Un point d'observation est défini par des instructions d'extraction de variable(s) à un point de programme donné. Cependant, dans le cas d'une simulation de plusieurs heures, certaines variables changent constamment, ce qui augmente considérablement la taille de la trace.

Par conséquent, il est également utile d'améliorer l'efficacité de la méthode de vérification. [15] propose une méthode pour vérifier des traces d'applications parallèles. Les propriétés sont vérifiées en utilisant des processus parallèles (un pour chaque trace), au lieu de fusionner toutes les traces en une seule. Cela limite le champ d'application de cette méthode aux programmes parallèles. Si fragmenter la génération de la trace n'est pas possible, on peut découper la trace après sa génération pour analyser chaque sous-trace indépendamment les unes des autres. C'est ce que nous proposons dans cet article.

2.2 Le contexte ferroviaire

De nos jours, chaque nouvelle ligne au sein de l'Union Européenne doit nécessairement satisfaire les règles nationales et les règles de spécification ERTMS/ETCS [5]. Ces règles sont une proposition de l'Union Européenne au problème d'interopérabilité entre les différents pays, pour les systèmes embarqués vis-à-vis des infrastructures au sol. Ce nouvel environnement légal et technologique doit être implémenté dans tous les pays membres. En effet, chaque règle d'une nouvelle ligne ferroviaire satisfait la norme ERTMS/ETCS et les règles nationales.

Les logiciels considérés dans ce contexte sont critiques et leur vérification est essentielle pour éviter les collisions ou presque collisions [2]. Pour ce faire, une plateforme propriétaire permet de simuler les comportements des trains. Cette plateforme virtualise le centre de contrôle, les lignes avec les gares, balises, des trains et les communications avec le centre de contrôle. Un train peut être conduit par un opérateur comme un train réel, ou défini par le scénario. Nous proposons d'adapter et de paralléliser l'approche proposée dans [13] afin d'analyser des propriétés temporelles sur les traces d'exécution issues de simulations menées sur cette plateforme.

3 Approche de base

L'approche définie dans [13] est résumée dans cette section. Ce travail consistait à vérifier des propriétés temporelles sur des logiciels avioniques embarqués.

3.1 Un langage adapté au contexte industriel

Plutôt que d'utiliser un langage complexe préexistant, nous avons préféré définir un langage adapté à notre contexte. La première étape du travail a donc été d'étudier plusieurs logiciels avioniques pour en extraire les propriétés à vérifier. Le langage défini permet d'exprimer un grand nombre de propriétés temporelles (y compris des propriétés de durée et de séquence). Ce langage s'est révélé adapté au contexte ferroviaire également.

Les propriétés sont donc formalisées en utilisant une combinaison de
- LTL,
- expressions régulières,
- opérateurs numériques (entiers et flottants) : comparaison entre numériques, addition, soustraction, multiplication, division,
- opérateurs spécifiques ci-dessous :
  - $x?T$ est l'instant où x a été modifié

5. *European Railway Transport Management System/European Train Control System*

— $x?C$ est le nombre de modifications subi par la variable
— $x?n$ est la valeur de la variable n modifications auparavant,
— $x?n?T$ est l'instant où variable x a été modifiée n fois auparavant,
— $x?n?C$ est la valeur du compteur de modification, n modifications auparavant.

3.2 Génération de la trace

Les traces à analyser sont issues de simulations. L'approche propose une technique pour générer les points d'observation nécessaires à la vérification de la propriété considérée. Ces points d'observation sont définis comme tous les points de programme pour lesquels une variable intervenant dans la propriété est modifiée.

L'analyse statique a servi à collecter l'ensemble de ces points d'observation. Un plugin de Frama-C [7] appelé Breakpointer a été implémenté pour cette tâche. Il repose sur le plugin Frama-C Value Analysis, qui effectue une analyse sémantique de programme, et donc tient compte des alias de variables.

Dans ce papier, nous adaptons cette technique pour le simulateur ERTMS qui permet de générer les traces. Chaque trace du simulateur est sauvegardée dans une base de données. Une trace correspond à une table contenant des états, i.e. l'instant, un type de message et une valeur associée. Le type de message décrit les variables modifiées.

3.3 Vérification

La phase de vérification se fait en deux temps :

1. Transformation de la propriété temporelle en automate de Büchi via *Ltl2ba*
2. Vérification de la propriété par exécution synchrone de l'automate de Büchi sur la trace.

Lorsque la propriété suit un patron prédéfini, des informations statistiques sont calculées par un automate statistique déterministe exécuté en même temps que l'automate de Büchi. Pour cet automate, chaque transition possède une étiquette contenant un prédicat, et une liste d'opérations sur des compteurs. Les opérations sont effectuées sur les compteurs lorsque le prédicat est vrai. Les compteurs conservent les informations statistiques et sont fournis à l'opérateur en fin de vérification.

4 Rappels

4.1 Notations utilisées

Dans les sections suivantes seront utilisées les notations classiques pour la formalisation. Le sens de ces notations est rappelé ci-dessous :

— $\mathcal{P}(Q)$ est l'ensemble des parties de Q ;
— par soucis de clarté, nous allons adopter la convention suivante. Si E est un produit cartésien d'ensembles tel que $E = E_1 \times E_2 \times ... \times E_n$, alors $E_{|i}$ est la projection de E sur E_i. En d'autres termes, $E_{|i} = E_i$. Si E_i est un produit cartésien lui même, tel que $E_i = E_{i,1} \times ... \times E_{i,m}$, alors $E_{|i,j}$ fait référence à $E_{i,j}$. Cette convention est également applicable aux fonctions ; $f : X_1 \times ... \times X_n \to Y_1 \times ... \times Y_m$ est une fonction. $f_{|i}$ est la projection de f sur Y_i ;
— Si E est un ensemble, E^ω est le produit cartésien infini de E ;
— $[\![a; b]\!]$, tel que $(a, b) \in \mathbb{N}^2$ et $a < b$, est l'ensemble des entiers $\{a, a + 1, \ldots, b\}$;
— Une trace σ est une séquence d'états. Un état de trace $\sigma_i \in \Sigma$, tel que Σ est l'alphabet, à l'index $i \in [\![0; |\sigma| - 1]\!]$ de la trace σ est une fonction totale qui retourne pour chaque variable sa valeur. $|\sigma|$ est la taille de la trace σ. Dans ce papier, **une trace est considérée comme un mot** basé sur l'alphabet Σ, dans le sens de la théorie des langages [22].

On définit aussi des nouvelles notations :

— si $\mathbb{K}$ est un ensemble de valeurs, alors $\mathbb{K}^{¿} = \mathbb{K} \cup \{¿\}$, où ¿ est la valeur *inconnue* ;

— par la suite, pour éviter les confusions, on parlera d'un élément pour un état de trace, et d'un état, pour un état d'automate.

4.2 Rappel des définitions d'un automate de Büchi [6]

Dans cette section est rappelée la définition formelle d'un automate de Büchi. La définition formelle d'un automate de Büchi statistique est également proposée. Ensuite, l'exécution d'un automate de Büchi sera schématisée.

4.2.1 Automate de Büchi

La définition de l'automate de Büchi s'appuie sur celle d'un automate classique :

Définition 1 *Un automate est classiquement défini comme un 5-uplet* $A = (Q, \Sigma, \rightarrow, q_0, F)$ *tel que :*

— Q *est un ensemble d'état ;*
— Σ *est un alphabet ;*
— $\rightarrow \subseteq Q \times \Sigma \times Q$ *est une relation de transition ;*
— $q_0 \in Q$ *est l'état initial ;*
— $F \subseteq Q$ *est un ensemble d'états finaux.*

La condition d'acceptation d'un mot fini par un automate est donnée par la définition suivante :

Définition 2 *Un mot fini* $w \in \Sigma^*$ *est reconnu par un automate* $A = (Q, \Sigma, \rightarrow, q_0, F)$*, si et seulement s'il existe une séquence* $(q)_{i \in [\![0;|w|]\!]}$*, qui commence à l'état initial* q_0*, tel que* $i \in [\![0; n-1]\!], (q_i, w_i, q_{i+1}) \in \rightarrow$ *et tel que* $q_n \in F$*.*

Un mot w *reconnu par* A *est écrit* $w \in \mathcal{L}(A)$*, où* $\mathcal{L}(A)$ *est l'ensemble des mots reconnus par* A*.*

Un automate de Büchi est un automate classique (définition 4) dont la condition d'acceptation permet de manipuler les mots/traces infinies. On parlera d'états *acceptants* plutôt que d'états finaux. La condition d'acceptation d'une trace est définie par :

Définition 3 *Un mot infini* $w \in \Sigma^\omega$ *est un mot de* $\mathcal{L}(B)$*, avec* $B = (Q, \Sigma, \rightarrow, q_0, F)$ *l'automate de Büchi, si et seulement si :*

— *il existe une séquence* $(q)_{i \in \mathbb{N}}$ *telle que pour tout* $i \in \mathbb{N}, (q_i, w_i, q_{i+1}) \in \rightarrow$ *;*
— *pour tout* $j \in \mathbb{N}$*, il existe* $k \in \mathbb{N}$ *tel que* $k > j$ *et* $q_k \in F$*.*

Dans ces travaux, une propriété est vérifiée sur une trace d'exécution. L'automate de Büchi correspondant est donc exécuté sur cette trace. L'alphabet utilisé est ainsi bâti sur les éléments.

Pour calculer des informations statistiques sur un mot infini donné, un automate de Büchi statistique est défini comme une extension d'un automate de Büchi déterministe. Lorsqu'une formule d'une transition donnée est vraie, des opérations primaires sont effectuées sur des compteurs donnés. À la fin de l'exécution de l'automate statistique, les compteurs quantifient des propriétés orthogonales à la propriété temporelle.

Exemple 1 *La propriété* $\Box (\Diamond\, e)$ *signifie que* e *se produit infiniment souvent. Le nombre d'occurrence de* e *est une information statistique.*

Dans cet article, l'automate de Büchi statistique remplace l'automate de Büchi classique lorsqu'une propriété temporelle correspond au patron de propriété de l'automate de Büchi statistique, contrairement à ce qui a été fait dans [13].

La première étape consiste à définir une opération statistique :

Définition 4 *Soit* $\mathcal{C}$ *un ensemble de variables entières appelées compteurs. L'ensemble des opérations statistiques* $\Lambda_{\mathcal{C}} : (\mathcal{C} \rightarrow \mathbb{Z}^\delta) \rightarrow (\mathcal{C} \rightarrow \mathbb{Z}^\delta)$ *est une liste d'action sur* $\mathcal{C}$*, dépendant de la valeur courante de tous les compteurs. Les opérations peuvent être :*

— *ne rien faire,*
— *affecter à un compteur une constante, la valeur d'un compteur, ou une expression numérique :*
  — *addition, soustraction,multiplication,division de compteurs/constantes*
  — *minimum, maximum de compteurs/constantes*

En cas d'exécution parallèle d'un automate de Büchi statistique déterministe, la valeur inconnue est nécessaire pour le calcul symbolique des compteurs.

Par définition, l'opération statistique λ, pour une transition donnée, est une liste d'actions.

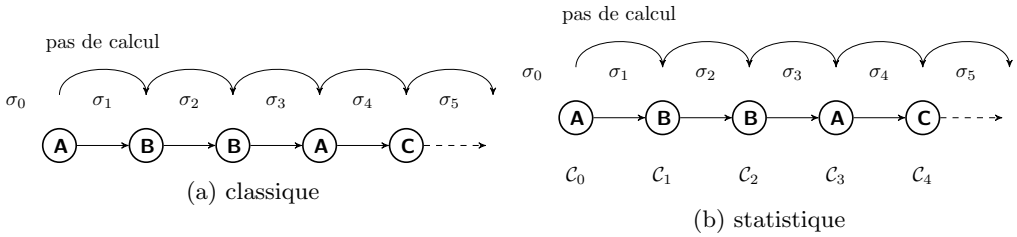
Enfin, l'automate de Büchi statistique est défini comme suit :

Définition 5 *Un automate de Büchi statistique est un 6-uplet $\mathcal{A} = (Q, \Sigma, \rightarrow, q_0, F, \mathcal{C}_0)$ tel que :*
— *Q est un ensemble d'états ;*
— *Σ est un alphabet ;*
— *$\rightarrow \subseteq Q \times \Sigma \times \Lambda_{\mathcal{C}} \times Q$ est une relation de transition ;*
— *$q_0 \in Q$ est un état initial ;*
— *$F \subseteq Q$ est un ensemble d'états acceptants ;*
— *$\mathcal{C}_0 : \mathcal{C} \rightarrow \mathbb{Z}^{\mathcal{C}}$, est une fonction retournant la valeur initiale de chaque élément de $\mathcal{C}$.*

Par la suite, le terme *mot*$\in \Sigma^*$ est remplacé par trace d'exécution.

4.2.2 Exécution d'un automate de Büchi : trois cas étudiés

On utilisera par la suite trois cas d'automate de Büchi : déterministe, déterministe statistique et non déterministe. Avant de proposer une définition formelle d'une exécution d'automate pour chacun de ces cas, nous proposons une illustration dans la figure 1 de ces cas sur un exemple.



(a) classique

(b) statistique

FIGURE 1 – Exécution d'un automate de Büchi déterministe

Un pas de calcul correspond à la consommation par l'automate de Büchi d'un élément pour activer les transitions accessibles. Lorsque l'automate de Büchi est déterministe (figure 1a), pour chaque pas de calcul, il n'y a qu'un état avant la consommation de l'élément courant et au plus un état après la consommation de cet élément. De ce fait, l'exécution d'un automate de Büchi peut être vue comme un chemin à plusieurs étapes (nœuds). Chaque nœud est un pas de calcul, et il n'y a qu'un seul lien entre deux nœuds.

L'exécution d'un automate de Büchi statistique déterministe correspond à l'exécution d'un automate de Büchi déterministe avec un ensemble de compteurs et leur valeur pour chaque état courant. Dans la figure 1b, les compteurs et leurs valeurs sont représentés par les $\mathcal{C}_i$.

Si l'automate de Büchi n'est pas déterministe, alors il y aura un ensemble d'états courants à la place d'un seul état. Le chemin est alors comparable à un treillis plutôt qu'à une séquence d'états. Dans la figure 2, chaque rectangle est l'ensemble des états courants après une étape de calcul. Le rectangle du niveau 1 est l'état initial, ensemble $\{A\}$. Celui du niveau 2 est $\{A, B\}$ après consommation du premier élément...

Schématiquement, un pas de calcul consiste à se déplacer d'un rectangle de niveau i à celui de niveau $i + 1$. Ainsi, la définition formelle d'une exécution de programme doit contenir deux ensembles d'états : un en début de pas de calcul, et un en fin de pas de calcul. Par la suite, cela sera pris en compte pour la formalisation d'une exécution.

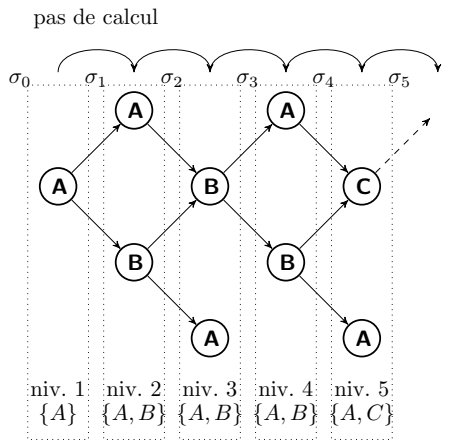


FIGURE 2 – Exécution d'un automate de Büchi non déterministe

L'exécution d'un automate de Büchi statistique n'est pas traitée ici. Par conséquent, dans la suite du papier, nous parlerons d'automate statistique pour un automate de Büchi statistique déterministe.

5 Exécution classique d'un automate de Büchi sur une trace

Cette section propose une formalisation de l'exécution séquentielle d'un automate de Büchi.

5.1 Exécution d'un automate non déterministe sur une trace

La formalisation de l'exécution d'un automate non déterministe servira de support aux formalisations proposées pour les autres cas. Dans l'approche de [13], les expressions régulières sont traduites en automates de Büchi déterministes avec un outil propriétaire AIRBUS, tandis que les formules LTL sont traduites en automates non déterministe par *Ltl2ba*.

Pour rappel, la définition 3 requiert une suite d'état, par conséquent, elle suffit pour traiter et formaliser complètement le premier cas. Cependant, dans le second cas, les automates de Büchi générés sont non déterministes. Pour le traitement de ce cas, la définition 3 est suffisante, néanmoins la formalisation complète nécessite la définition d'une suite non pas d'états mais d'ensembles d'états.

La formalisation d'une exécution d'un automate de Büchi non déterministe sur une trace repose sur la figure 2. Une exécution d'automate est une suite de pas de calcul. Chaque pas est une paire d'ensembles d'états : un ensemble avant la consommation d'un élément, et un après. Ainsi, nous définissons formellement l'exécution d'un automate de Büchi non déterministe de la manière suivante :

Définition 6 *Un mot infini $w \in \Sigma^\omega$ est un mot de $\mathcal{L}(B_{nd})$, où $B_{nd} = (Q, \Sigma, \rightarrow, q_0, F)$ est un automate de Büchi non déterministe, si et seulement si existe une suite $R_{i\in\mathbb{N}} \in \mathcal{P}(Q)^2$ tel que :*

- $R_0 = (\{q_0\}, \{q_0\}) \in \mathcal{P}(Q)^2$ (1)
- $\forall i \in \mathbb{N}^*$, $R_i \in \mathcal{P}(Q)^2$, *tel que :*
  - $R_{i|1} = R_{i-1|2}$ (2)
  - $\forall r_i \in R_{i|2}, \exists r_{i-1} \in R_{i|1}$, *tel que* $(r_{i-1}, w_i, r_i) \in \rightarrow$ (3)
- $\forall j \in \mathbb{N}$, $\exists k \in \mathbb{N}$ *tel que* $k > j$ *et* $\exists R_{k|1} \cap F \neq \emptyset$. (4)

La propriété 2 assure la consistance de l'exécution de l'automate : l'ensemble d'états courants en fin d'un pas de calcul est identique à l'ensemble d'états courants au début du pas de calcul

suivant. La propriété 3 est une partie de la définition d'un mot accepté par l'automate, traitant de l'existance d'un chemin pour un automate de Büchi donné. La propriété 4 est l'autre partie de la définition d'un mot accepté par l'automate, qui stipule l'existence d'un état acceptant infiniment souvent accessible.

Si un mot infini w n'est pas reconnu par un automate de Büchi non déterministe, alors soit il existe k tel que $R_{k|2} = \emptyset$ et pour tout $k' > k, R_{k'} = (\emptyset, \emptyset)$, soit la suite d'ensembles d'états courants contient un nombre fini d'états acceptants.

Pour la figure 2, on obtient la suite : $R_0 = (\{A\}, \{A\})$; $R_1 = (\{A\}, \{A, B\})$; $R_2 = (\{A, B\}, \{A, B\})$; $R_3 = (\{A, B\}, \{A, B\})$; $R_4 = (\{A, B\}, \{A, C\})$.

Dans notre contexte, les traces finies manipulées sont transformées en traces infinies par bouclage sur le dernier élément afin de permettre l'utilisation d'outils tel que *Ltl2ba* pour transformer des propriétés LTL en automate de Büchi, sans modification particulière de la sémantique de LTL. De plus, le problème de fin de trace n'est pas abordé ici, il a été traité dans [13].

La limite de cette méthode d'exécution est de devoir calculer séquentiellement la suite de pas de calcul $R_{i \in [\![0;|\sigma|-1]\!]}$.

5.2 Exécution d'un automate statistique sur une trace

La formalisation de l'exécution d'un automate statistique sur une trace est la suivante :

Définition 7 *Un mot infini $w \in \Sigma^\omega$ est un mot de $\mathcal{L}(B_S)$, où $B_S = (Q, \Sigma, \rightarrow, q_0, F, \mathcal{C}_0)$, si et seulement si il existe une suite $R^S_{i \in \mathbb{N}} \in (Q)^2 \times (\mathcal{C} \longrightarrow \mathbb{Z}^\delta)^2$ telle que :*

- $R^S_0 = (q_0, q_0, \mathcal{C}_0, \mathcal{C}_0)$ (5)
- $\forall n \in \mathbb{N}^*$, $R^S_n \in (Q)^2 \times (\mathcal{C} \longrightarrow \mathbb{Z}^\delta)^2$ *tel que :*
  - $R^S_{n|1} = R^S_{n-1|2}$ *et* $R^S_{n|3} = R^S_{n-1|4}$ (6)
  - $(R^S_{n|1}, w_i, \lambda_n, R^S_{n|2}) \in \rightarrow$ (7)
  - $R^S_{n|4} = \lambda_n(R^S_{n|3})$
- $\forall j \in \mathbb{N}$, $\exists k \in \mathbb{N}$ *tel que* $k > j$ *et* $R^S_{k|1} \in F$. (8)

Les propriétés 6 assurent la consistance de l'exécution de l'automate. La propriété 7 est une partie de la définition d'un mot accepté par l'automate. La propriété 8 est l'autre partie de la définition d'un mot accepté par l'automate, qui stipule qu'un état acceptant est infiniment souvent atteint. Dans la propriété 7, w_n est l'élément courant et λ_n l'opération courante appliquée aux compteurs.

Si on reprend la figure 1b, alors la suite sera : $R^S_0 = (A, A, \mathcal{C}_0, \mathcal{C}_0)$; $R^S_1 = (A, B, \mathcal{C}_0, \mathcal{C}_1)$; $R^S_2 = (B, B, \mathcal{C}_1, \mathcal{C}_2)$; $R^S_3 = (B, A, \mathcal{C}_2, \mathcal{C}_3)$; $R^S_4 = (A, C, \mathcal{C}_3, \mathcal{C}_4)$.

6 Exécution parallélisée d'un automate de Büchi sur une trace

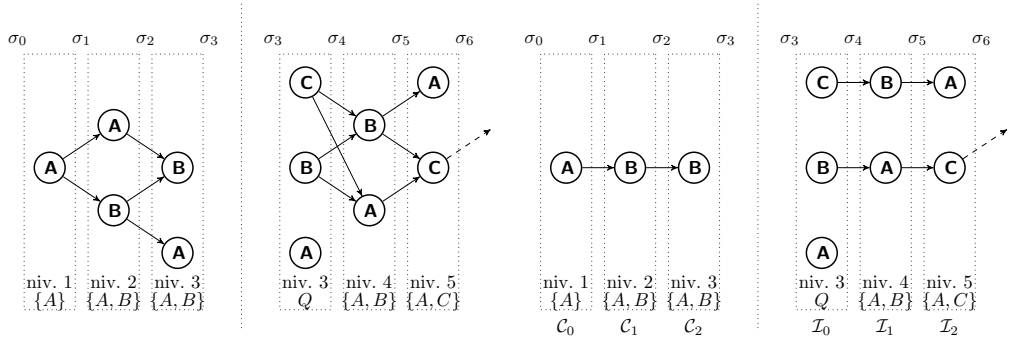
Nous arrivons maintenant au cœur de la problématique. Après en avoir expliqué les principes de l'approche, nous l'appliquons sur un automate de Büchi non déterministe, puis sur un automate statistique. Nous apportons ensuite la preuve de l'équivalence entre l'analyse séquentielle et parallèle d'une trace et évoquons quelques limites.

6.1 Principe

Jusqu'à présent, lors de l'analyse d'une trace, il était nécessaire de calculer la suite de pas de calcul $R_0, ..., R_{i-1}$, avant de pouvoir calculer le pas R_i. Nous proposons l'approche suivante :

1. découper la trace en plusieurs sous traces,
2. exécuter l'automate de Büchi B sur chaque sous trace indépendamment les unes des autres,
3. fusionner les résultats d'exécution intermédiaires pour déterminer si la trace est dans $\mathcal{L}(B)$.

La figure 3 schématise l'approche sur une trace divisée en deux, à l'état 3. L'automate de Büchi est exécuté sur les sous-traces $\sigma_0 \dots \sigma_2$ et $\sigma_3 \dots \sigma_5$. L'exécution de l'automate de Büchi sur la première trace suit la formalisation de la section précédente. L'exécution de l'automate de Büchi sur la seconde trace commence avec Q ($\{A, B, C\}$) comme ensemble d'états initiaux.



(a) non déterministe (b) statistique

FIGURE 3 – Principe de l'approche parallèle

La vérification de la propriété sur la trace entière est effectuée par l'utilisation d'une fonction agissant comme un raccourci entre le début et la fin de la seconde sous-trace.

Par exemple, si la trace se termine à l'état 5, alors on voit sur la figure 3a que :

— l'état A avant l'élément 3 ne conduit à rien après l'élément 5,

— les états B et C avant l'élément 3 conduisent tous les deux à A et C après l'élément 5.

Comme au niveau 3, l'ensemble des états courants est $\{A, B\}$, alors l'ensemble des états après l'élément 5 est $\{A, C\}$.

Après avoir illustré ce principe, nous allons le formaliser pour un automate de Büchi non déterministe et pour un automate statistique. Par souci de compréhension, la formalisation se fera sur une trace divisée en deux sous-traces, mais elle est généralisable à plusieurs divisions.

6.2 Exécution d'un automate de Büchi non déterministe

La trace σ est coupée en deux à l'état d'indice c. Il est cependant nécessaire que l'élément d'indice c soit complet. En d'autres termes, il faut que la valeur de chaque variable soit définie dans cet état, même si celle-ci n'est pas modifiée. Il suffit alors de définir une fonction pour cet état statuant sur la modification d'une variable ou non à cet état σ_c. L'automate sera exécuté sur chacune des deux sous-traces. Pour ce faire, deux suites de pas de calculs sont définies : $\mathcal{R}_{n\in[\![0;c]\!]}$ et $\overline{\mathcal{R}}_{n\in[\![0;|\sigma|-c+1]\!]}$. $\mathcal{R}_{n\in[\![0;c]\!]}$ est la suite de pas de calcul débutant à l'élément σ_0 comme définie dans la section 5. Ainsi, pour tout $n \in [\![0; c]\!]$, $\mathcal{R}_n = R_n$.

La suite de pas de calcul sur la seconde trace est définie de la façon suivante :

Définition 8 *La suite* $\overline{\mathcal{R}}_{n\in[\![0;|\sigma|-c+1]\!]} \in (\mathcal{P}(Q))^2 \times (Q \longrightarrow \mathcal{P}(Q))^2$ *est telle que :*

— *La définition des deux premiers composants de* $\overline{\mathcal{R}}_n$ *est la suivante :*

— $\overline{\mathcal{R}}_{0|1} = Q$ (9)

— $\forall n \in [\![0; |\sigma| + 1 - c]\!], \overline{\mathcal{R}}_{n|1} = \overline{\mathcal{R}}_{n-1|2}$. (10)

— $\forall n \in [\![0; |\sigma| + 1 - c]\!]$, $\forall r'_n \in \overline{\mathcal{R}}_{n|2}$, $\exists r_n \in \overline{\mathcal{R}}_{n|1}$, *tel que* $(r_n, \sigma_{c+n+1}, r'_n) \in \rightarrow$ (11)

— *La définition des deux derniers composants de* $\overline{\mathcal{R}}_n$ *correspond à la fonction* α *définie ci-après :*

— $\forall n \in [\![0; |\sigma| + 1 - c]\!], \overline{\mathcal{R}}_{n|3} = \alpha_n$, (12)

— $\forall n \in [\![1; |\sigma| + 1 - c]\!], \overline{\mathcal{R}}_{n|4} = \overline{\mathcal{R}}_{n+1|3}$, (13)

— $\forall q \in Q$, $\alpha_0(q) = \{q\}$. (14)

— $\alpha_n(q) = \{q'' | \exists q' \in \alpha_{n-1}(q), (q'\sigma_{c+n+1}, q'') \in \rightarrow\}$. (15)

Concrètement, $\overline{\mathcal{R}}$ est la suite de pas de calcul de l'automate débutant à l'élément d'index $c+1$ et finissant à l'élément d'index $|\sigma|-1$. L'ensemble d'états initiaux est Q. En effet, l'exécution de l'automate sur chaque sous-trace étant indépendante, on ne connaît donc pas l'ensemble des états courants de l'automate en début de la seconde sous-trace. C'est l'opération de fusion qui permet de recalculer a posteriori cette information, grâce à la fonction α_n. En effet, celle-ci sauvegarde le lien entre l'ensemble des états à l'élément $c+1$ et l'ensemble des états à l'élément $|\sigma|-1$.

La propriété 9 signifie que la suite de pas de calcul est initialisée avec Q comme ensemble d'états initiaux. Les propriétés 10 et 11 assurent respectivement la consistance de l'exécution et l'acceptation d'une trace par l'automate. α est initialisée par la fonction identité (propriété 14). Pour chaque état, α est calculée récursivement en fonction des transitions activées (propriété 15).

Si on reprend l'exemple de la figure 3a, alors la suite sera : $\mathcal{R}_0 = (\{A\}, \{A\})$; $\mathcal{R}_1 = (\{A\}, \{A, B\})$; $\mathcal{R}_2 = (\{A, B\}, \{A, B\})$; $\overline{\mathcal{R}}_0 = (\{A, B, C\}, \{A, B\}, \alpha = id, \{\alpha(A) = \emptyset, \alpha(B) = \{A, B\}, \alpha(C) = \{A, B\}\})$, $\overline{\mathcal{R}}_1 = (\{A, B\}, \{A, C\}, \{\alpha(A) = \emptyset, \alpha(B) = \{A, B\}, \alpha(C) = \{A, B\}\}, \{\alpha(A) = \emptyset, \alpha(B) = \{A, C\}, \alpha(C) = \{A, C\}\})$.

L'exécution de l'automate de Büchi sur chacune des deux sous-traces données est indépendante. Les résultats de l'analyse de chaque sous-trace doivent être assemblés après, grâce à la fonction α.

Pour effectuer une exécution parallélisée d'un automate de Büchi en plus de deux sous-traces, la démarche à suivre est la même que pour deux sous-traces. En effet, le préfixe de la trace est analysé normalement et pour chaque autre sous-trace on utilise les définitions de $\overline{\mathcal{R}}$.

6.3 Exécution d'un automate statistique

Soit σ la trace, $B_S = (Q, \Sigma, \rightarrow, q_0, F, \mathcal{C}_0)$ est un automate de Büchi statistique déterministe.

La suite de pas de calcul ressemble à $\overline{\mathcal{R}}$ définie dans la section 6.2. Une fonction additionnelle est calculée comme la fonction α, pour construire les opérations appliquées aux compteurs entre le début et la fin de chaque sous-trace.

$\mathcal{R}^S_{n\in[\![0;c]\!]}$ est la suite de pas de calcul débutant à l'élément σ_0 comme défini dans la section 5.2. Ainsi, pour tout $n \in [\![0; c]\!]$, $\mathcal{R}^S_n = R^S_n$.

L'autre suite de pas de calcul est construite comme celle de la section précédente, en tenant compte du calcul des informations statistiques :

Définition 9 $\overline{\mathcal{R}}^S_{n\in[\![0;|\sigma|+1-c]\!]} \in (\mathcal{P}(Q))^2 \times (Q \rightarrow \mathcal{P}(Q))^2 \times (Q \rightarrow \Lambda_{\mathcal{C}})^2$*, où, pour tout* $n \in [\![0; |\sigma| + 1 - c]\!]$ *:*

- $\overline{\mathcal{R}}^S_{n|1} = \overline{\mathcal{R}}_{n|1}$, $\overline{\mathcal{R}}^S_{n|2} = \overline{\mathcal{R}}_{n|2}$
- $\overline{\mathcal{R}}^S_{n|3}$ *(fonction* α*) est proche de* $\overline{\mathcal{R}}_{n|3}$*. La dernière équation à propos de* α *définie à la section 6.2 est la seule qui change :*
  - $\forall q \in Q, \alpha_n(q) = \bigcup q''$ *tel que* $\exists q \in \alpha_{n-1}, (q', \sigma_{c+n+1}, \lambda_{n,q'}, q'') \in \rightarrow$
  - *la propriété* $\overline{\mathcal{R}}^S_{n|4} = \overline{\mathcal{R}}^S_{n+1|3}$ *est préservée*
- $\forall q \in Q, \overline{\mathcal{R}}^S_{0|5}(q) = Inconnu.$
- $\forall q \in Q, \overline{\mathcal{R}}^S_{n|6}(q) = \lambda_{(n,q')}(\overline{\mathcal{R}}^S_{n|5}(q))$.
- $\forall q \in Q, \overline{\mathcal{R}}^S_{n|6}(q) = \overline{\mathcal{R}}^S_{n+1|5}(q)$.

Si on reprend la figure 3b, en ajoutant les informations statistiques, alors la suite sera :
$\mathcal{R}_0 = (\{A\}, \{A\}, \mathcal{C}_0, \mathcal{C}_0)$; $\mathcal{R}_1 = (\{A\}, \{A, B\}, \mathcal{C}_0, \mathcal{C}_1)$; $\mathcal{R}_2 = (\{A, B\}, \{A, B\}, \mathcal{C}_1, \mathcal{C}_2)$;
$\overline{\mathcal{R}}_0 = (\{A, B, C\}, \{A, B\}, \alpha = id, \{\alpha(A) = \emptyset, \alpha(B) = \{A\}, \alpha(C) = \{B\}\}, \mathcal{I}_0, \mathcal{I}_1)$;
$\overline{\mathcal{R}}_1 = (\{A, B\}, \{A, C\}, \{\alpha(A) = \emptyset, \alpha(B) = \{A\}, \alpha(C) = \{B\}\}, \{\alpha(A) = \emptyset, \alpha(B) = \{C\}, \alpha(C) = \{A\}\}, \mathcal{I}_1, \mathcal{I}_2)$. Si la transition de X à Y provoque le calcul statistique $\lambda_{X,Y}$, alors :
$\mathcal{I}_2(A) = Inconnu$,
$\mathcal{I}_2(B) = \lambda_{A,C}(\mathcal{I}_1(B)) = \lambda_{A,C}(\lambda_{B,A}(Inconnu))$
$\mathcal{I}_2(C) = \lambda_{B,A}(\mathcal{I}_1(C)) = \lambda_{B,A}(\lambda_{C,B}(Inconnu))\}$

6.4 L'opération de fusion

L'opération de fusion consiste à rassembler les informations calculées sur les deux sous traces. L'ensemble d'états courants obtenu après l'exécution de l'automate sur la première sous trace se combine avec les informations de la fonction α calculée pendant l'exécution de l'automate sur la seconde sous-trace.

Soit $\mathcal{F}$, l'ensemble des états de l'automate à l'élément $|\sigma| - 1$. Alors $\mathcal{F} = \{q \in \overline{\mathcal{R}}_{|\sigma|-1|2}$, tel que $\exists q' \in \mathcal{R}_{c|2}$ et $q \in \alpha_n(q')\}$.

Théorème 1 *La vérification de la propriété par exécution séquentielle d'un automate de Büchi est équivalent à la vérification de cette propriété par exécution parallélisée de l'automate de Büchi.*

Si la trace est découpée en plus de deux sous-traces, il suffit d'assembler les résultats séquentiellement.

Preuve 1 *Par construction, la fonction α calcule un raccourci d'exécution : pour chaque état de $\overline{\mathcal{R}}_{0|2}$ à l'élément σ_c, on a l'ensemble des états qu'il produit $\overline{\mathcal{R}}_{|\sigma||1}$ en fin d'exécution. L'opération de fusion consiste à déterminer les éléments de $\overline{\mathcal{R}}_{|\sigma||1}$ à conserver. Les éléments à conserver sont ceux que génère l'ensemble $\mathcal{R}_c$ par application de la fonction α. Par définition de $\mathcal{F}$, on a donc $\mathcal{F} = R_{|\sigma||1}$*

De ce fait, une propriété peut être vérifiée en utilisant l'algorithme de fin de trace défini dans [13] et $\mathcal{F}$.

6.5 Limitations

Dans [13], des opérateurs spécifiques sur des variables ont été définis pour calculer à la volée des compteurs de modification de ces variables, ou obtenir une valeur antérieure à un nombre de modifications donné. Des exemples sont donnés dans la section 3.1.

Comme ces éléments sont calculés à la volée, il est impossible d'utiliser la méthode de vérification parallèle. Pour contourner ce problème, on pourrait ajouter des nouvelles variables dans la trace pour obtenir ces informations directement.

7 Résultats expérimentaux

Des résultats expérimentaux sont présentés dans cette section. Les essais sont réalisés sur des traces génériques pour les raisons évoquées dans la section 1. Le but de ces expériences est d'évaluer l'efficacité de la nouvelle approche et de la comparer, en terme de temps d'exécution, avec l'approche séquentielle.

7.1 Implémentation d'un prototype

Pour des raisons de confidentialité, il est impossible de fournir le code source du programme implémentant l'approche de ce papier. Cependant, certains éléments techniques sont fournis. Le prototype est écrit en C++ et comporte 32900 lignes de code, dont 13000 lignes de fichier *header*.

7.1.1 Une architecture distribuée

Les deux approches (séquentielles et parallèles) ont été conçues sur une architecture distribuée, en vue de partager efficacement les ressources matérielles disponibles. Ainsi, l'opération de lecture de la trace est séparée de l'exécution de l'automate de Büchi. La figure 4 illustre la relation entre les différents processus de l'approche séquentielle. Le processus d'exécution et le processus de lecture communiquent via une interface TCP/IP. Le processus d'exécution charge les informations à la volée d'un ou plusieurs processus de lecture. En effet, chaque processus peut lire une sous-trace différente simultanément avec les autres processus. La figure 5, quant à elle, schématise les relations de communication entre les différents processus de l'approche parallèle ; néanmoins, l'opération de

fusion n'apparaît pas. Chaque processus d'exécution peut interagir avec un ou plusieurs processus de lecture.

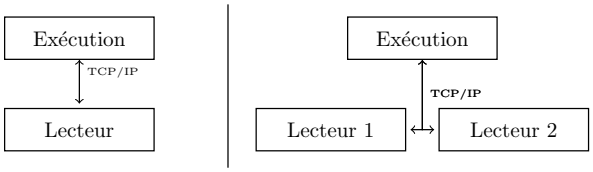


FIGURE 4 – Implémentation séquentielle

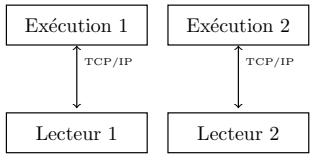


FIGURE 5 – Implémentation parallèle

7.1.2 Le format de trace

Un format de trace basé sur Xml a été implémenté. La trace est découpée en plusieurs blocs qui contiennent le même nombre d'éléments $|b|$. Ainsi, lorsque le processus d'exécution a besoin d'un élément σ_i, il appelle le processus de lecture qui retourne le bloc complet contenant σ_i. Ce bloc σ_i contient donc la sous-trace $\sigma_i ... \sigma_{i+|b|-1}$. Les blocs sont utilisés pour deux raisons. D'une part, cela limite les problèmes de latence dus au réseau. Des essais ont montré que le temps de vérification augmentait considérablement si les éléments étaient envoyés les uns après les autres. D'autre part, les sous-traces de la version parallélisée suivent la segmentation par bloc. En effet, les traces utilisées sont partielles [6], mais les éléments σ_i et $\sigma_{i+|b|-1}$ sont complets [7] pour chaque bloc b commençant à σ_i.

7.2 Expériences de charge

Deux expériences sont faites sans information statistique, et avec informations statistiques. Les essais sont menés sur la même trace générique pour les deux expériences. Il s'agit d'une trace à 10 millions d'éléments et ne contenant qu'une variable appelée x. La valeur de la variable change à chaque élément. Le domaine de variation de la variable est $[\![-10; 10]\!]$. Pour effectuer une vérification sur plusieurs tailles de trace, le processus peut être arrêté après un nombre donné d'états. Ainsi, on peut vérifier un préfixe de la trace pour simuler des traces plus petites.

La figure 7 rassemble les résultats pour les deux expériences. La trace a été découpée en 2, 4, 8 ou 10 sous-traces. La vérification a été effectuée sur chaque sous-trace et le temps maximum pour chaque classe de division a été placé dans ce graphique. Le processus de lecture a été lancé sur un ordinateur de 24Go-RAM, tandis que le processus d'exécution a été exécuté sur un ordinateur de 4Go-RAM avec un processeur Core-5i (2Ghz, 3Mo de cache, 64-bits).

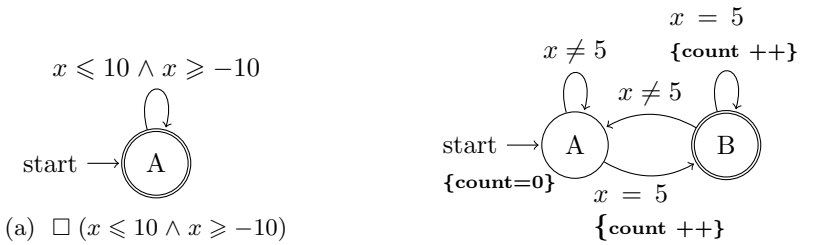
Sans information statistique La propriété à vérifier sur cette trace est $\Box\ (x \leqslant 10 \wedge x \geqslant -10)$. Cela signifie que la contrainte $x \leqslant 10 \wedge x \geqslant -10$ est vraie pour tout élément de la trace. Le but de cet essai est de comparer les temps de vérification entre l'approche séquentielle et l'approche parallèle lorsqu'il n'y a pas d'information statistique

L'automate de Büchi de cette propriété obtenue avec *Ltl2ba*, est représenté en figure 6a.

En comparaison avec le temps de vérification séquentiel, le temps de vérification est réduit de 40% si la trace est divisée en deux sous-traces. Si la trace est divisée en 10 sous-traces, le temps de vérification est réduit de 90%. Le temps de fusion reste négligeable, à moins de 0.01 seconde lorsque la trace est divisée en 10.

Avec information statistique La propriété étudiée est $\Box\ (\Diamond\ (x = 5))$. Elle signifie que la contrainte $x = 5$ se produit infiniment souvent. Pendant l'analyse de la trace, le nombre d'éléments où la propriété $x = 5$ survient est calculé à l'aide du compteur *count*, dont la valeur initiale est 0.

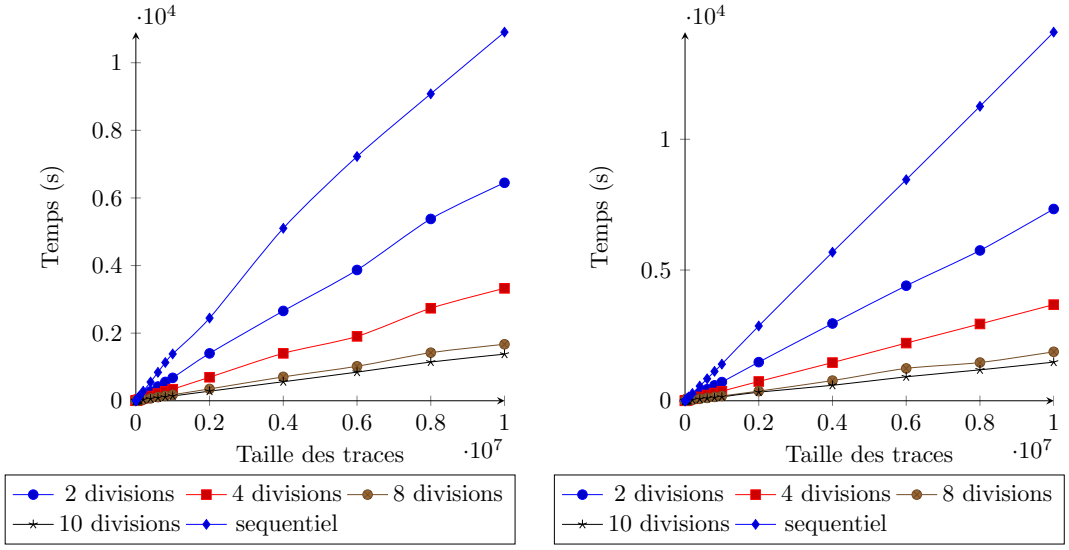
6. Un élément ne contient que les variables modifiées
7. Toutes les variables et leur valeurs courantes sont dans cet état.



(a) $\square\ (x \leqslant 10 \wedge x \geqslant -10)$

(b) $\square\ (\Diamond\ (x = 5))$

FIGURE 6 – Automates de Büchi des expériences



(a) Sans information statistique

(b) Avec information statistique

FIGURE 7 – Temps maximum requis pour la vérification d'une trace

La figure 7b montre que la vérification de la propriété est plus efficace lorsque la trace est découpée, que lorsque la trace est analysée séquentiellement. Le temps de vérification est environ divisé par deux lorsque la trace est divisée par deux, divisé par 4 lorsque la trace est divisée par 4... Le temps de fusion reste négligeable, moins de 0.01 seconde lorsque la trace est divisée en 10.

Cette approche parallèle est donc plus efficace que l'approche séquentielle. L'opération de fusion des résultats intermédiaires est négligeable pour dix divisions. La prochaine étape de ces travaux doit consister à analyser une trace d'exécution provenant d'un cas réel industriel, avec des propriétés plus complexes et des traces contenant plus de variables. En effet, les automates de Büchi générés par des propriétés plus complexes, ont généralement plus d'états et de transitions que ceux des expériences menées ci-dessus.

8 Conclusion

Dans cet article, nous avons formalisé l'exécution sur une trace d'automates de Büchi de type non-déterministe, déterministe ou statistique. Ces formalisations reposent sur la définition formelle d'un automate de Büchi et les conditions d'acceptation d'un mot, qui ne décrivent pas précisément chaque pas de calcul. Cette formalisation est un socle pour la contribution de ce papier qui consiste à paralléliser l'exécution d'un automate de Büchi sur une trace.

La trace analysée est découpée en deux sous-traces, puis l'automate de Büchi est exécuté sur

chacune d'entre elles. L'exécution sur la première sous-trace est classique tandis que sur l'autre elle varie. En effet, l'automate de Büchi est initialisé avec l'ensemble des états de l'automate au lieu de l'état initial. Pendant l'exécution de l'automate sur la seconde sous-trace, un raccourci entre un état au premier élément de la sous-trace et l'ensemble des états qu'il génère au dernier élément de la sous-trace est calculé. Ce raccourci est ensuite utilisé pour fusionner les résultats d'analyse obtenus entre la première et la seconde sous-trace et permet de déduire le résultat global.

Cette nouvelle approche a été appliquée sur des traces génériques de différentes tailles. Le temps de vérification de l'approche parallèle a été comparé à celui de l'approche séquentielle. Le temps de vérification est divisé environ par le nombre de coupures effectuées sur la trace. Cette nouvelle approche limite donc le temps de vérification et la mémoire requise pour analyser une trace, puisque l'analyse indépendante des sous-traces est possible. La prochaine étape de ces travaux consistera à analyser des traces d'exécution issues de cas industriels réels, et d'effectuer les comparaisons entre l'approche séquentielle et l'approche parallèle.

Un léger inconvénient de cette nouvelle approche est que les opérateurs spécifiques développés en [13] ne peuvent pas être employés, puisqu'ils induisent un calcul à la volée non parallélisable. Malgré tout, nous avons montré que le temps de vérification reste meilleur avec l'approche parallèle. Cette approche inclut également le calcul d'informations statistiques sur une trace.

Une des perspectives de ces travaux pourrait être d'enrichir la trace avec les informations nécessaires à l'utilisation des opérateurs spécifiques. Une autre piste de recherche serait de déterminer le nombre de coupures de trace maximum permettant un gain de temps de calcul.

Remerciement

Ce travail est financé par l'ANR dans le contexte du projet PERFECT. Ce projet est également supporté par le pôle de compétitivité national ITRANS.

Références

[1] J.-R. Abrial. The B-book : Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA, 1996.

[2] F. Aguirre, M. Sallak, W. Schon, and F. Belmonte. Application of evidential networks in quantitative analysis of railway accidents. Proceedings of the Institution of Mechanical Engineers, Part O, Journal of Risk and Reliability, 227(4) :368–384, Nov 2013.

[3] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Eagle does space efficient ltl monitoring. Technical report, Nasa, 2003.

[4] Shay Berkovich, Borzoo Bonakdarpour, and Sebastian Fischmeister. Gpu-based runtime verification. In Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, IPDPS '13, pages 1025–1036, Washington, DC, USA, 2013. IEEE Computer Society.

[5] E. Bodden. Efficient hybrid typestate analysis by determining continuation-equivalent states. In Software Engineering, 2010 ACM/IEEE 32nd International Conference on, volume 1, pages 5–14, May 2010.

[6] J.Richard Büchi. On a decision method in restricted second order arithmetic. In Saunders Mac Lane and Dirk Siefkes, editors, The Collected Works of J. Richard Büchi, pages 425–435. Springer New York, 1990.

[7] CEA-LIST and INRIA-Saclay. The frama-c platform for static analysis of c programs, 2008.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst., 8(2) :244–263, April 1986.

[9] Patrick Cousot and Radhia Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1977.

[10] Marcelo d'Amorim and Grigore Rosu. Efficient monitoring of omega-languages. In CAV'05, pages 364–378, 2005.

[11] Doron Drusinsky. The temporal rover and the atg rover. In K. Havelund, J. Penix, and W. Visser, editors, SPIN Model Checking and Software Verification, volume 1885 of Lecture Notes in Computer Science. Springer, 2000.

[12] A. Ferlin and V. Wiels. Combination of static and dynamic analyses for the certification of avionics software. In Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on, pages 331–336, Nov.

[13] Antoine Ferlin. Vérification de propriétés temporelles sur des logiciels avioniques par analyse dynamique formelle. PhD thesis, Institut Supérieur de l'Aeronautique et de l'Espace (ISAE), Université de Toulouse, ED-MITT, September 2013.

[14] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01), volume 2102 of LNCS. Springer, 2001.

[15] Markus Geimer, Felix Wolf, BrianJ.N. Wylie, and Bernd Mohr. Scalable parallel trace-based performance analysis. In Bernd Mohr, JesperLarsson Träff, Joachim Worringen, and Jack Dongarra, editors, Recent Advances in Parallel Virtual Machine and Message Passing Interface, volume 4192 of Lecture Notes in Computer Science, pages 303–312. Springer Berlin Heidelberg, 2006.

[16] D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In Automated Software Engineering, 2001.

[17] K. Havelund and G. Rosu. Monitoring programs using rewriting. In Automated Software Engineering, pages 135 – 143, 2001.

[18] Klaus Havelund and Kestrel Technology. A rewriting-based approach to trace analysis. Automated Software Engineering, 12 :2005, 2002.

[19] Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. An overview of the mop runtime verification framework. International Journal on Software Tools for Technology Transfer, 14 :249–289, 2012.

[20] R. Pellizzoni, P. Meredith, M. Caccamo, and G. Rosu. Hardware runtime monitoring for dependable cots-based real-time embedded systems. In Real-Time Systems Symposium, 2008, pages 481–491, Nov 2008.

[21] A. Pnueli and A. Zaks. Psl model checking and run-time verification via testers. In FM 2006 : Formal Methods, volume 4085 of LNCS. Springer, 2006.

[22] Grzegorz Rozenberg and Arto Salomaa, editors. Handbook of Formal Languages, Vol. 1 : Word, Language, Grammar. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[23] Volker Stolz and Eric Bodden. Temporal assertions using aspectj. Electronic Notes in Theoretical Computer Science, 144(4) :109 – 124, 2006. Proceedings of the Fifth Workshop on Runtime Verification (RV 2005).

[24] Haitao Zhu, M.B. Dwyer, and S. Goddard. Predictable runtime monitoring. In Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on, pages 173–183, July 2009.

[25] C.B. Zilles and G.S. Sohi. A programmable co-processor for profiling. In High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on, pages 241–252, 2001.