Actes des 22ème Journées des approches formelles dans l'assistance au développement de logiciels,AFADL'23

Natalia Kushik, Frédéric Mallet

▶ **To cite this version:**

Natalia Kushik, Frédéric Mallet. Actes des 22ème Journées des approches formelles dans l'assistance au développement de logiciels,AFADL'23. pp.60, 2023, Journées AFADL. hal-04179353v2

HAL Id: hal-04179353
https://inria.hal.science/hal-04179353v2

Submitted on 1 Sep 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

AFADL 2023

actes des 22ème journées des

Approches Formelles dans l'Assistance au Développement de Logiciels

conjointement avec les journées du GDR GPL

édités par

Natalia Kushik and Frédéric Mallet

Rennes

du 5 au 8 juin 2023

Contents

Préface

Les journées AFADL ont pour objectif de rassembler de nombreux acteurs académiques et industriels intéressés par la mise en œuvre des techniques formelles aux divers stades du développement des logiciels et/ou des systèmes. Elles ont pour but de mettre en valeur les travaux récents effectués autour de thèmes comme :

- Les techniques et outils formels contribuant à assurer un bon niveau de confiance dans la construction de logiciels et de systèmes ;
- Les méthodes et processus permettant d'exploiter efficacement les techniques et outils formels disponibles ou conçus ;
- Les méthodes et processus mettant en œuvre des techniques formelles différentes et hétérogènes dans un développement ;
- Les leçons tirées de la mise en œuvre de ces outils ou principes sur des études de cas ou des applications industrielles.

Les techniques et outils présentés assistent notamment les activités suivantes : la modélisation, la validation et la gestion d'exigences formelles applicables aux logiciels, les spécialisations ou extensions de techniques de modélisation et d'évaluation induites par des domaines applicatifs (télécommunication, contrôle-commande, robotiques, systèmes interactifs, architectures, composition de services, applications distribuées sur le web...) ou des points de vue particuliers sur les systèmes (sécurité informatique, exécution temps réel...), le passage d'une étape de conception à la suivante : patrons de raffinement de spécifications, déploiement d'une architecture logicielle sur une architecture matérielle, génération automatique de code, réutilisation de composants, le test et l'évaluation rigoureuse de modèles formels ou codes, la spécification et la vérification formelles d'architectures, de modèles et de programmes.

On s'intéresse aussi à la combinaison d'approches formelles avec des approches informelles ou semi-formelles ou à la coopération de techniques formelles de développement avec des techniques plus classiques (par exemple à la complémentarité vérification formelle / test pour les aspects V&V), aussi bien qu'à l'adaptation des approches formelles aux techniques d'apprentissage automatique et au domaine de l'informatique quantique.

En 2023, les journées AFADL se sont tenues à Rennes conjointement avec les journées du GDR GPL. Il y a eu deux sessions communes avec le GT MTV2 et le GT LVP. Il y avait plus de 50 participants en présentiels.

Comité de programme

Président(e)s

- Natalia Kushik, SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris
- Frédéric Mallet, Université Côte d'Azur, CNRS, Inria, I3S

Relecteurs

- Akram Idani, LIG, Grenoble INP
- Alain Giorgetti, Université de Franche-Comté
- Antoine Rollet, LaBRI, Université de Bordeaux, Bordeaux INP
- Aurélie Hurault, IRIT-ENSEEIHT, Toulouse
- Benoit Valiron, LMF, CentraleSupélec
- Catherine Dubois, ENSIIE-Samovar, Paris
- Celia Picard, ENAC
- Christian Attiogbé, LS2N, Nantes Université
- David Deharbe, CLEARSY
- David Delahaye, LIRMM, Université Montpellier, CNRS, Montpellier
- David Monniaux, VERIMAG, CNRS
- Delphine Longuet, Thales Research & Technology
- Fatiha Zaidi, LMF, Université Paris Saclay
- Ileana Ober, IRIT, Université Toulouse 3
- Ioannis Parissis, Grenoble INP, Université Grenoble Alpes
- Jean-Baptiste Raclet, IRIT, Université Toulouse III
- Julien Signoles, CEA-LIST, Université Paris-Saclay
- Laurent Voisin, Systerel
- Ludovic Henrio, LIP Laboratory, ENS Lyon

- Marc Frappier, Université de Sherbrooke
- Marc Pouzet, DIENS, ENS Paris
- Mathieu Jaume, LIP6 Sorbonne Université
- Mehdi Lhommeau, ISTIA, Université d'Angers
- Micaela Mayero, LIPN, Université Sorbonne Paris Nord
- Mohammed Foughali, IRIF, Université de Paris
- Nikolai Kosmatov, Thales TRT, CEA List
- Olivier Barais, IRISA
- Olivier Hermant, Mines Paris
- Pascal Poizat, LIP6, Université Paris Nanterre
- Pascale Le Gall, MICS, CentraleSupélec
- Pierre-Cyrille Heam, FEMTO-ST, Université de Franche-Comté, CNRS
- Rabea Ameur-Boulifa, LabSoc, Télécom Paris
- Régine Laleau, Université Paris-Est Créteil
- Romain Rouvoy, Cristal, Université de Lille
- Sandrine Blazy, Irisa, Université de Rennes
- Stephan Merz, Inria Nancy
- Tewfik Ziadi, LIP6, Sorbonne Université
- Thomas Lambolais, LGI2P,Mines Alès
- Virginie Wiels, ONERA / DTIS
- Vlad Rusu, Inria
- Yamine Aït Ameur, IRIT, ENSEEIHT

Une approche pour inférer les expressions de calcul géométrique en modélisation à base topologique

Romain Pascual[1], Pascale Le Gall[1], Hakim Belhaouari[2], and Agnès Arnould[2]

[1]MICS, CentraleSupélec, Université Paris-Saclay
[2]Université de Poitiers, Univ. Limoges, CNRS, XLIM, Poitiers

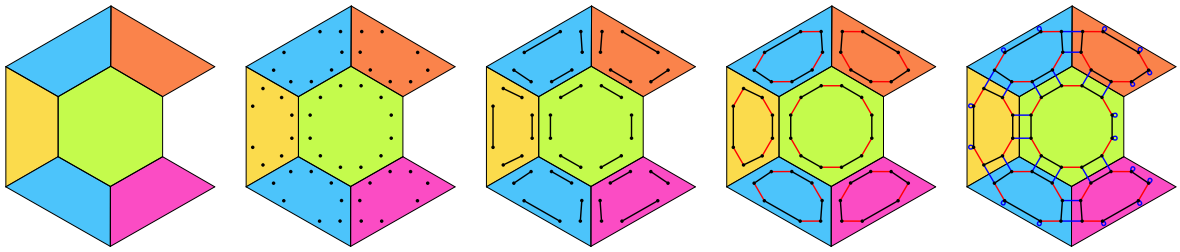
Résumé

La conception d'opérations de modélisation géométrique repose sur leur implantation dans un langage de programmation. Même si cette tâche peut être simplifiée en exploitant une description de haut niveau de ces opérations, la difficulté de les implanter contraste avec l'apparente simplicité de la description d'une opération sur un exemple. Nous proposons une méthode d'inférence d'opérations à partir d'un exemple représentatif. Plus précisément, nous nous plaçons dans le domaine de la modélisation géométrique à base topologique qui propose une représentation d'objets nD par décomposition en cellules topologiques (sommets, arêtes, faces, volumes, etc.) sur lesquelles sont ajoutées des informations géométriques. Ce domaine admet une spécification de la topologie par des structures combinatoires qui peuvent être représentées à l'aide de graphes, de sorte qu'une opération se formalise comme une règle de transformation de graphes. Dans cet article, nous complétons notre algorithme d'inférence du calcul topologique avec une approche pour la déduction des expressions de calcul géométrique. Notre approche exploite deux idées principales : des abstractions topologiques des expressions géométriques pour assurer la généricité des calculs retrouvés et une représentation comme un problème de satisfaction de contraintes de l'expression recherchée.

Mots-clés : modélisation géométrique à base topologique ; inférence d'opérations ; synthèse de code ; langage dédié ; CSP ; règles de transformations de graphes.

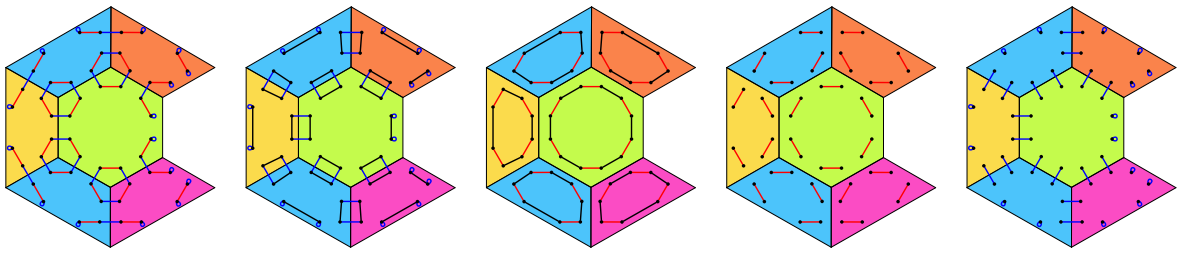
1 Introduction

Concevoir une nouvelle opération de modélisation géométrique suppose généralement l'élaboration d'un algorithme qui décrit les modifications à effectuer et l'implantation de l'algorithme dans un langage de programmation efficace. Nous proposons une approche exploitant l'intuition que l'on peut avoir de l'illustration d'une opération sur un exemple pertinent. Plus précisément, nous travaillons dans le domaine de la modélisation géométrique à base topologique qui sépare les modifications topologiques et géométriques de l'opération. Nous réalisons l'inférence d'opération en exploitant le langage dédié de la plateforme Jerboa [2] qui exploite les cartes généralisées comme représentation des objets et les règles de transformations de graphes comme formalisation des opérations. L'approche décrite ici permet de compléter l'algorithme de repliement topologique [7] en étudiant l'inférence d'expressions de calcul pour les informations dites de plongement qui encodent la géométrie de l'objet. L'inférence d'expressions géométriques associée à l'inférence topologique ainsi qu'à la compilation des règles permet ainsi la synthèse de code pour des opérations de modélisation géométrique à partir d'un simple exemple représentatif d'utilisation. Nous étudierons principalement les calculs géométriques liés aux positions des sommets pour lesquels nous supposerons que les nouvelles positions géométriques sont obtenues par combinaisons affines de barycentres topologiques.



(a) (b) (c) (d) (e)

Figure 1: G-carte associée à un objet géométrique: (a) objet géométrique 2D, (b) brins (points noirs), (c) ajout des 0-arcs (en noir), (d) ajout des 1-arcs (en rouge), (e) ajout des 2-arcs (en bleu).



(a) (b) (c) (d) (e)

Figure 2: Cellules topologiques et orbites: (a) orbites $\langle 1, 2 \rangle$ (sommets), (b) orbites $\langle 0, 2 \rangle$ (arêtes), (c) orbites $\langle 0, 1 \rangle$ (faces), (d) orbites $\langle 1 \rangle$ (sommets de faces), (e) orbites $\langle 2 \rangle$ (sommets d'arêtes).

2 Représentation des objets et leur manipulation dans un langage dédié

2.1 Représentation topologique et géométrique d'un objet nD

Nous représentons les objets géométriques avec le modèle des cartes généralisées [3] vues sous forme de graphes [8].

Une *carte généralisée* de dimension n (avec $n \geq 0$), n-G-carte ou simplement G-carte, est un graphe non orienté, étiqueté sur les arcs par les dimensions de $0..n$ (où $i..j$ est l'ensemble des entiers entre i et j inclus) et vérifiant deux conditions de consistance topologique:

- *contrainte d'arcs incidents* : tout nœud possède un unique arc incident par dimension.
- *contrainte de cycles* : pour toutes dimensions d et d' telles que $d \leq d' + 2$, tout chemin étiqueté par $dd'dd'$ est un cycle.

Par héritage de la vision combinatoire, on appelle *brin* d'une G-carte les nœuds du graphe. Un brin représente un n-uplet de cellules topologiques. Ainsi, un brin d'une 2-G-carte représente un sommet, une arête et une face. Les arcs du graphes expliquent alors comment les cellules topologiques sont reliées entre elles. Plus précisément, un arc de dimension d explicite que deux cellules de dimension d sont adjacentes tout en partageant les cellules des autres dimensions. Ainsi, un arc de dimension 0, aussi appelé 0-arc, sépare deux sommets d'une même arête et d'une même face.

A partir de ces informations, on peut reconstruire la G-carte associée à un objet, comme illustré sur la Figure 1. Partant d'un objet 2D (Figure 1a), on ajoute un brin pour chaque triplet (sommet, arête, face) valide (Figure 1b). On relie ensuite les paires de brins séparant les sommets d'une même arête vue d'une même face à l'aide de 0-arcs (en noir sur la Figure 1c). Cette construction est ensuite itérée par dimension croissante, on ajoute des 1-arcs entre les brins d'un même sommet et d'une même face mais d'arêtes différentes (en rouge sur la Figure 1d) et finalement des 2-arcs pour relier les faces le long d'une arête commune (en bleu sur la Figure 1e). On notera qu'un brin du bord topologique de dimension d est relié à lui-même par un d-arc. Par exemple, les faces du bord engendrent des 2-boucles sur la Figure 1e.

Cette représentation permet de définir les cellules topologiques comme des sous-graphes induits par un ensemble de dimensions. Etant donné un brin v d'une G-carte G et un ensemble de dimensions

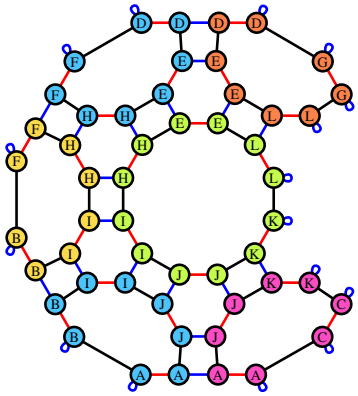


Figure 3: G-carte plongée par l'ajout des positions et des couleurs (représentés dans les brins).

$o \subset 0..n$, l'*orbite* $G\langle o\rangle$(v) est le sous-graphe maximal induit par o et contenant v. On dit alors que $G\langle o\rangle$(v) est de type $\langle o\rangle$ ou est une $\langle o\rangle$-orbite. Par exemple, l'orbite $G\langle 1,2\rangle(v)$ contient l'ensemble des brins auquel on peut accéder en n'empruntant que des arcs de dimension 1 et 2 à partir du brin v. Vu autrement, les $\langle 1,2\rangle$-orbites sont les composantes connexes du graphe construites à partir de la G-carte en supprimant les arcs de dimension 0. Ces orbites encodent les sommets de l'objet, comme illustré sur la Figure 2a. De même, on encode les arêtes avec les $\langle 0,2\rangle$-orbites (Figure 2b) et les faces avec les $\langle 0,1\rangle$-orbites (Figure 2c). Les orbites décrivent aussi des graphes qui ne sont pas des cellules topologiques. Par exemple, la Figure 2d illustre les $\langle 1\rangle$-orbites qui encodent les sommets de faces et la Figure 2e les $\langle 2\rangle$-orbites qui encodent les sommets d'arêtes. D'autres orbites ont été illustrées implicitement lors de la construction de la G-carte comme les orbites vides de type $\langle\rangle$, réduites à un brin (Figure 1b), les $\langle 0\rangle$-orbites (Figure 1c), qui encodent les arêtes de faces, ou les orbites de type $\langle 0,1,2\rangle$, qui encodent les composantes connexes, ici l'objet complet (Figure 1e).

Les informations géométriques ajoutées sur cette structure topologique sont encodées par des fonctions de plongement qui associent une valeur d'un type donné aux cellules topologiques. Ainsi, nous pouvons ajouter des positions aux sommets ou des couleurs aux faces. Comme les cellules topologiques ne sont pas des éléments atomiques de la G-carte mais des sous-graphes, un plongement π est décrit par un type d'orbite $\langle o\rangle$ et un type de donnée τ (au sens des types abstraits algébriques). On obtient alors des fonctions de plongement π de profil $\langle o_\pi\rangle \rightarrow \tau_\pi$. Par exemple, les positions sont encodées par une fonction `position` : $\langle 1,2\rangle \rightarrow$ `Point3` et les couleurs par une fonction `color` : $\langle 0,1\rangle \rightarrow$ `ColorRGB` où `Point3` et `ColorRGB` désignent les types de données usuels utilisés en informatique graphique. Formellement, nous utilisons la description par des graphes attribués [4] qui permet de manipuler les valeurs attachées aux nœuds du graphe [6]. Dans notre cas, chaque nœud de la G-carte possède une unique valeur pour chaque plongement avec la condition que tous les brins d'une $\langle o_\pi\rangle$-orbite possède la même valeur pour le plongement π. L'ajout des plongements à l'objet de la Figure 1 est illustré en Figure 3 où les informations de position et de couleur sont visibles sur chacun des brins de la G-carte.

2.2 Un langage fondé sur les transformations de graphes

Nous exploitons une formalisation des opérations de modélisation géométrique sous la forme de règles de réécriture de graphes [4, 6]. De telles règles sont une extension aux structures non-linéaires de la réécriture de mots ou de termes. Une règle se présente donc sous la forme $L \rightarrow R$ où L et R sont des graphes, décrivant respectivement le motif à modifier et le motif réécrit. Pour des questions de généricité, nos règles se présentent sous la forme de schémas de règles paramétrés par un type d'orbite $\langle o\rangle$ afin d'abstraire les changements topologiques. Ces schémas de règles sont au cœur de la conception d'opérations dans la plateforme Jerboa [2]. Un exemple de schéma de règles est donné en Figure 4a, illustrant la triangulation barycentrique d'une face. Ce schéma de règles modifie une $\langle 0,1\rangle$-orbite, c'est-à-dire une face. Plus précisément, le nœud étiqueté $\langle 0,1\rangle$ du membre gauche de la règle est appelé à être instancié par une face avec un nombre arbitraire de brins. Les différents nœuds

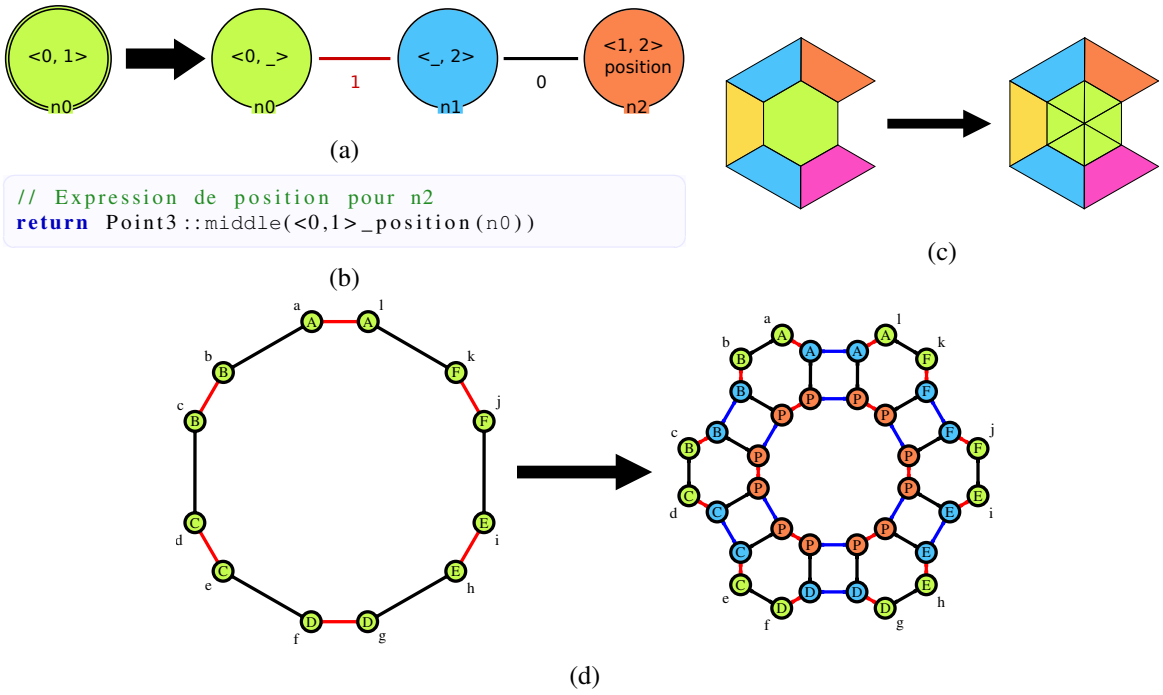


Figure 4: Schéma de règles pour la triangulation d'une face: (a) schéma de règles, et (b) expression géométrique associée au nœud $n2$.

du motif droit décrivent alors les différentes orbites à modifier ou à ajouter avec les modifications topologiques à effectuer. Par exemple, le nœud $n0$ possède le type d'orbite $\langle 0, \_\rangle$. Les brins et les 0-arcs de la face filtrée sont préservés tandis que ses 1-arcs sont supprimés, ce qui déconnecte les arêtes de la face. De même, le nœud $n2$ possède le type d'orbite $\langle 1, 2\rangle$ pour construire le sommet dual de la face, ajouté pour trianguler cette dernière : l'orbite d'origine est dupliquée en renommant les 0-ars en 1-arcs et les 1-arcs en 2-arcs. Le nœud $n1$ et les deux arcs du membre droit permettent de connecter la face initiale au sommet dual. L'instanciation du schéma de règles sur une face hexagonale est donnée en Figure 4d où la couleur des brins décrit l'association avec le nœud du schéma de la Figure 4a: par exemple les brins $\{a, b, c, d, e, f, g, h, i, j, k, l\}$ sont associés au nœud $n0$.

L'annotation `position` sur le nœud $n2$ à droite indique qu'il porte une expression géométrique. Le calcul correspondant est donné en Figure 4b. L'expression renvoie un terme de type `Point3` dont la valeur est donnée par l'expression $\texttt{middle}(\langle 0, 1\rangle\texttt{\_position}(n0))$. Dans cette expression, `middle` retourne le barycentre d'une collection de positions et $\langle 0, 1\rangle$`_position` exprime que l'on doit récupérer l'ensemble des positions dans une orbite $\langle 0, 1\rangle$. Lors de l'application de la règle, l'identifiant $n0$ sera substitué par les brins associés au nœud $n0$ avant modification. Dans le cas présent, il s'agit des brins de la face que l'on triangule. En substituant $n0$ par l'identifiant d'un des brins de la face, par exemple b, on obtient alors $\texttt{middle}(\langle 0, 1\rangle\texttt{\_position}(b))$ qui encode le barycentre des positions des sommets de l'orbite $G\langle 0, 1\rangle(b)$, c'est-à-dire le barycentre de la face. Soulignons que n'importe quel choix parmi les brins $a, \ldots, l$ encoderait le même barycentre.

3 Inférence d'opérations

Dans cet article, nous nous intéressons à l'inférence d'opérations de modélisation géométrique. Tout comme la description des objets et la formalisation du langage, cette inférence est nécessairement double, c'est-à-dire topologique et géométrique. La reconstruction des modifications topologiques a été étudiée dans [7] et repose sur un algorithme de repliement. Il s'agit intuitivement d'un parcours de graphe qui reconstruit le schéma de règles à partir d'un exemple représentatif. Un tel exemple consiste en une instance de l'objet avant modification, une instance de l'objet après modification, d'informations associant les parties non modifiées de l'objet et enfin d'un type d'orbite spécifiant la

généricité de l'opération. Nous présentons ici une méthode pour déduire les expressions de calcul des modifications géométriques à associer à la règle afin de compléter l'inférence topologique. Nous disposons toujours d'un exemple représentatif de l'opération, du résultat de l'algorithme de l'inférence topologique [7] augmentés de résultats intermédiaires obtenus lors des calculs de repliements.

3.1 Espace de recherche

Nous commençons par le cas des positions associées aux sommets, plongement nécessaire pour afficher des objets polyédriques. Puisque nous disposons d'une instance de l'objet avant et après modification, nous pouvons accéder aux valeurs de plongement des brins associés à un nœud du membre droit comme du membre gauche. En particulier, nous pourrons vérifier que le résultat du calcul de l'expression ajoutée à un nœud du membre droit de la règle correspond à la valeur de plongement des brins associés. Dans un premier temps, nous proposons une approche pour déterminer le placement des expressions géométriques sur les nœuds du membre droit :

1. Si v_R est un nœud préservé du membre droit et que tous les brins qui lui sont associés ont la même position dans les instances avant et après, alors aucun calcul ne doit être réalisé, et nous laissons l'expression du nœud droit vide. Nous marquons l'orbite $\langle 1,2\rangle(v_R)$ comme résolue.
2. Pour chaque $\langle 1,2\rangle$-orbite non résolue, nous choisissons un nœud référent pour accueillir une expression de plongement.

Le marquage des $\langle 1,2\rangle$-orbites dans l'étape 1 et le choix d'un référent par $\langle 1,2\rangle$-orbite non résolue dans l'étape 2 repose sur la propagation des valeurs de plongement calculées lors de l'application d'un schéma de règles. On réduit ainsi en amont le nombre d'expressions géométriques à résoudre. Dans le cas où nous ne pourrions pas trouver de solution pour le nœud référent choisi à l'étape 2, la contrainte de cohérence des plongements [1] assure que nous n'en trouverions pas pour les autres nœuds de son orbite puisque les brins ont nécessairement la même position.

Une fois que nous avons identifié les nœuds référents pour accueillir une expression de plongement, il faut retrouver une expression géométrique valide pour l'ensemble des brins associés aux nœuds. Plus encore, l'expression doit rester valide qu'importe l'instanciation. Autrement dit, le résultat du calcul ne doit pas dépendre du brin choisi pour effectuer le calcul et nous avons besoin d'une solution permettant d'encoder l'invariance du brin dans le calcul induite par l'abstraction topologique des schémas de règles. La solution que nous proposons est d'utiliser des opérateurs *adhoc* en lien avec les informations topologiques comme l'opérateur de collecte qui récupère l'ensemble des valeurs d'une orbite. De manière plus générale, nous appelons *point d'intérêt* toute solution qui encode une abstraction de la topologie sous-jacente invariante au choix du brin.

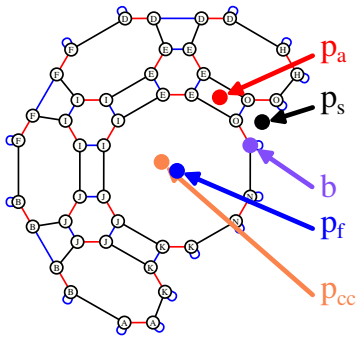
Bien que les points d'intérêt éliminent la question du choix du brin dans le calcul, il convient encore de spécifier l'ensemble des fonctions à considérer pour résoudre concrètement l'inférence des expressions géométriques. Nous appelons *famille de fonctions* cet ensemble de fonctions. Nous allons considérer la famille des combinaisons affines qui présentent le double avantage d'être souvent utilisées pour les opérations de modélisation géométrique tout en étant simples à résoudre. Concrètement, nous construisons, pour chacun des nœuds référents, une équation symbolique à partir des points d'intérêts. L'idée est ensuite d'imiter l'instanciation du schéma de règles en évaluant cette équation sur chaque brin associé au nœud de la règle. On remplace donc chaque point d'intérêt par sa valeur calculée en fonction des positions de l'instance avant modification, ce qui permet de générer un système d'équations affines. Ce système définit un problème de satisfaction de contraintes (ou CSP) où les variables sont les poids de la combinaison de domaine $\mathbb{R}$, liées par une unique contrainte déduite de l'évaluation de l'équation symbolique. Nous déléguons ensuite la résolution du CSP à un solveur dédié (nous avons expérimenté avec les solveurs OR-tools de Google et Z3 de Microsoft).

3.2 Points d'intérêt

Comme évoqué précédemment, une règle est symétrique par rapport aux orbites portées par ses nœuds. Ainsi, une expression de plongement doit permettre de calculer les positions de tous les

Point d'intérêt	Nom	Expression
sommet	p_s	middle($\langle 1,2\rangle$_position(b))
arête	p_a	middle($\langle 0,2\rangle$_position(b))
face	p_f	middle($\langle 0,1\rangle$_position(b))
cc	p_{cc}	middle($\langle 0,1,2\rangle$_position(b))

(a)



(b)

Figure 5: Points d'intérêt: (a) expressions des points d'intérêt , (b) calculs sur un objet.

brins instances du nœud tout en s'évaluant de manière identique pour tous les brins d'une même orbite de plongement. Pour assurer ces propriétés, nous exprimons les équations par des points d'intérêt invariants par symétrie sur les différentes orbites. Nous proposons d'utiliser les barycentres des positions des différentes orbites. En exploitant que le type d'orbite $\langle 1,2\rangle$ du plongement position induit une relation d'équivalence sur l'ensemble des orbites (dépendant également de la contrainte de cycle), on peut se ramener à considérer seulement 4 orbites pour un objet 2D. Ces orbites correspondent aux sommets, arêtes, faces et composantes connexes. Les barycentres d'une orbite $\langle o\rangle$ sont encodés par les expressions middle($\langle o\rangle$_position(v)) où v est un nœud gauche du schéma. Il s'agit de l'expression déjà introduite pour la triangulation barycentrique (cf. Section 2.2) mais en anonymisant l'orbite. Les points d'intérêt sont donnés dans le tableau 5a et illustrés en Figure 5b.

Nous généralisons ces expressions comme des expressions de plongement pour pouvoir les utiliser sur un schéma de règles quelconque et notons $\mathrm{PI}(v)$ pour l'ensemble des expressions des points d'intérêt du nœud v. A partir de ces points, on obtient donc l'équation symbolique suivante :

$$\texttt{position}(v_R) = \sum_{v_L \in V_L} \sum_{p \in \mathrm{PI}(v_L)} w_{p,v_L} p(v_L) + t \tag{1}$$

où V_L est l'ensemble des nœuds du membre gauche du schéma de règles, $(w_{p,v_L})_{v_L \in V_L}$ sont des poids (inconnus), et t encode une translation intrinsèque (inconnue).

3.3 Génération automatique du code des expressions géométriques

Nous générons automatiquement le code de l'expression géométrique à partir de la solution trouvée par le solveur CSP. Cette expression exploite le langage de script de Jerboa qui étend la vision purement algébrique des calculs [1] dans un langage impératif [5] dont la syntaxe est similaire au langage `Java`. Voici le modèle générique du code de l'expression déduite :

```
Point3 res = new Point3( #translation# );                              // translation

// pour tout noeud #node# du motif gauche
Point3 pS_#node# = Point3::middle(<1,2>_position( #node# ));           // sommet
pS_#node#.scaleVect( #w(s,#node#)# );                                  // poids
res.addVect(pS_#node#);

Point3 pA_#node# = Point3::middle(<0,2>_position( #node# ));           // arete
pA_#node#.scaleVect( #w(a,#node#)# );                                  // poids
res.addVect(pA_#node#);

Point3 pF_#node# = Point3::middle(<0,1>_position( #node# ));           // face
pF_#node#.scaleVect( #w(f,#node#)# );                                  // poids
res.addVect(pF_#node#);

Point3 pCC_#node# = Point3::middle(<0,1,2>_position( #node# ));        // composante connexe
pCC_#node#.scaleVect( #w(cc,#node#)# );                                // poids
res.addVect(pCC_#node#);

return res;
```

Les expressions générées permettent l'ajout des différents points d'intérêt au vecteur de translation. Les expressions `Point3::middle(< #orb# >_position( #node# ))` code le calcul des points d'intérêt qui sont ensuite multipliés par le poids calculé lors de la résolution du CSP via l'expression `#poi#.scaleVect( #w(#poi#,#node#)# )`. Finalement, la contribution du point d'intérêt est ajouté au calcul global avec l'expression `res.addVect(#poi#);`. Dans le cas où le poids calculé est nul (avec un seuil de tolérance), le code associé n'est pas généré.

Pour conclure, nous mentionnons aussi que l'on exploite la condition de cohérence des plongements pour réduire le nombre de variables du système. En effet, si deux nœuds sont dans la même $\langle 1, 2\rangle$-orbite du membre gauche, leurs brins associés auront la même position. On peut donc remplacer V_L par $V_L/\langle 1, 2\rangle$ dans l'équation 1.

3.4 Résolution de la triangulation barycentrique

Nous illustrons notre méthode en reprenant le schéma de règles pour la triangulation (cf. Figure 4a) et en considérant les instances avant et après de la face hexagonale (cf. Figure 4d). Le nœud $n0$ est préservé et les brins qui lui sont associés $(a, b, \ldots, l)$ possèdent les mêmes positions $(A, B, \ldots, F)$ avant et après modification. Nous pouvons en déduire que $n0$ est résolu sans avoir besoin d'expression. Nous marquons donc l'orbite $\langle 1, 2\rangle(n0)$ comme résolue, ce qui permet d'éliminer le nœud $n1$ (cf. étape 1). Il reste alors le nœud $n2$ à résoudre via l'équation symbolique 1. Puisque le membre gauche ne contient que le nœud $n0$, on obtient l'équation:

$$\texttt{position}(n2) = \underbrace{w_{(s,n0)}p_s(n0)}_{\text{sommet}} + \underbrace{w_{(a,n0)}p_a(n0)}_{\text{arête}} + \underbrace{w_{(f,n0)}p_f(n0)}_{\text{face}} + \underbrace{w_{(cc,n0)}p_{cc}(n0)}_{\text{composante connexe}} + t$$

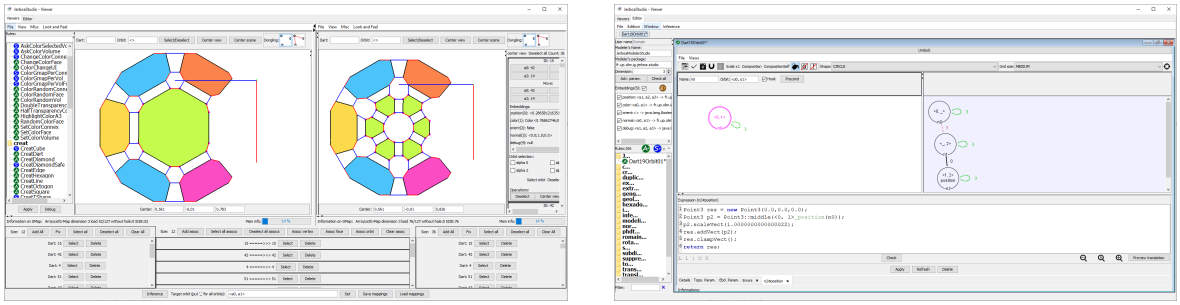
On évalue cette équation sur les 12 brins de l'hexagone (cf. Figure 4d). En projetant le système sur les axes x, y et z, nous obtenons un système à 36 équations et 7 variables. A l'aide d'un solveur, nous obtenons la solution $w_{(s,n0)} = 0.0$, $w_{(a,n0)} = 0.0$, $w_{(f,n0)} = 1.0$, $w_{(cc,n0)} = 0.0$, $tx = 0.0$, $ty = 0.0$, et $tz = 0.0$. A partir de cette solution, nous générons le code suivant :

```
Point3 res = new Point3(0.0,0.0,0.0);              // translation nulle
Point3 p2 = Point3::middle(<0,1>_position(n0));    // face
p2.scale(1.0);                                     // poids
res.addVect(p2);
return res;
```

3.5 Généralisation à un espace vectoriel

Bien que nous ayons expliqué l'inférence des expressions pour les positions associées aux sommets, cette approche fonctionne pour d'autres types de plongement. En effet, dès lors que les valeurs de plongement appartiennent à un espace vectoriel, notre méthode par combinaison affines de barycentre fonctionne. On étend la notion de points d'intérêt en *valeurs d'intérêt*, qui collectent les valeurs dans une orbite et calculent le barycentre du multi-ensemble obtenu. En pratique, il suffit d'encapsuler les plongements vectoriels en codant des opérateurs de somme et de multiplication par un scalaire d'une part et de généraliser le calcul des orbites pertinentes pour un plongement donné.

Nous avons ainsi expérimenté cette approche sur le type de données `ColorRGB` porté par le type d'orbite $\langle 0, 1\rangle$, c'est-à-dire les faces. La principale difficulté liée aux couleurs RGB provient de leur encodage comme des vecteurs dans $[0, 1]^3$ et non dans $\mathbb{R}^3$. Une première solution consiste à laisser l'expression déduite inchangée. L'application des règles peut alors conduire à des couleurs en dehors de l'espace représentable, ce qui convient lorsque l'utilisateur dispose d'un mécanisme pour traiter ces cas limites. Une deuxième solution est de modifier automatiquement la solution déduite afin de restreindre le résultat du calcul. Cette modification se produit sur l'expression déduite et ne modifie pas la résolution du système d'équations. Nous pouvons aussi retourner une couleur par défaut ou une couleur aléatoire comme solution finale.



(a) (b)

Figure 6: JerboaStudio : (a) viewer et (b) éditeur.

3.6 JerboaStudio

Nous avons implanté notre méthode dans la plateforme Jerboa [2]. Jerboa contient : (a) un éditeur qui permet la conception d'opérations sous forme de règles, (b) un noyau qui permet de traduire ces règles en code efficace et (c) un viewer qui permet d'afficher des objets géométriques représentés sous forme de G-cartes et de leur appliquer les opérations obtenues par synthèse de code à partir des règles. L'éditeur permet à un spécialiste de la modélisation géométrique de construire un logiciel pour un expert d'un domaine applicatif qui sera amené à utiliser uniquement le viewer. L'approche de conception d'opération par inférence à partir d'un exemple présentée ici pour les aspects géométriques et dans [7] pour les aspects topologiques vise à rendre caduque cette séparation des tâches et donc des différents composants de Jerboa. Nous avons donc conçu un outil entièrement intégré appelé JerboaStudio et illustré en Figure 6. Une version de JerboaStudio est disponible librement en ligne.[1]

4 Une application en 3D: l'éponge de Menger

L'éponge de Menger est une extension en 3D de l'ensemble de Cantor (1D) ou du tapis de Sierpinski (2D). Une étape de subdivision peut être construite comme suit : prendre un cube et diviser chaque face en 9 carrés pour obtenir 27 cubes, puis enlever tous les cubes du milieu (milieu des faces et centre du cube initial). L'itération suivante applique cette subdivision à chacun des 20 cubes restants. Afin de pouvoir inférer les éléments géométriques de cette opération, nous avons, dans un premier temps, construit l'éponge de Menger (Figure 7b) à partir du cube (Figure 7a) à l'aide d'une règle écrite par nos soins (Figure 7e). Il s'agit ici d'une opération volumique sur une 3-G-carte qui suppose que le schéma de règles associé utilise le type d'orbite $\langle 0, 1, 2, 3 \rangle$, utilisé ici pour annoter le nœud $n0$. Nous avons ensuite retiré les expressions géométriques de la règle (Figure 7e).

À partir des deux premiers objets (Figure 7a et 7b) et du schéma de règles épuré, nous avons appliqué la méthode d'inférence décrite dans cet article. Notons que chaque nœud du schéma de règles encode 48 brins (8 brins par face carrée, et 6 faces carrées pour un cube) de sorte qu'il ne manque que 3 calculs de positions. Ces expressions sont portées par les nœuds $n1$, $n7$, et $n16$ dans la Figure 7e. Des exemples de sommets associés à ces nœuds sont donnés en Figure 7f. Le brin de référence pour ces trois sommets appartient au sommet inférieur droit de l'objet. Cette démarche a été motivée par l'idée de pouvoir comparer la règle inférée avec la règle de référence construite manuellement. A titre d'illustration, nous donnons ci-dessous les expressions inférées pour les 3 nœuds $n1$, $n7$ et $n16$.

[1] Dépôt consulté le 22 mai 2023 : https://gitlab.com/jerboateam/jerboa-studio

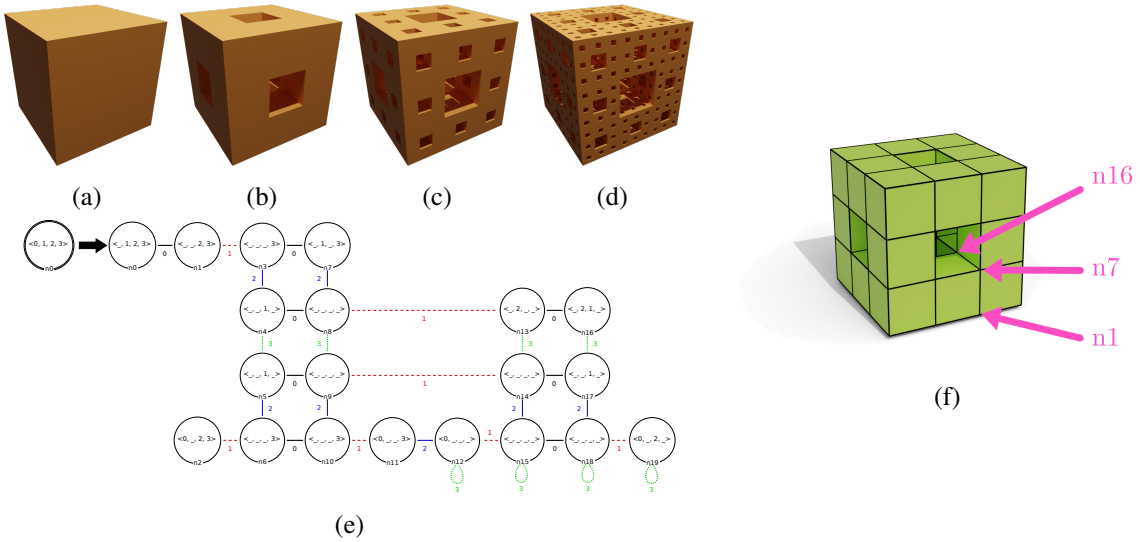


Figure 7: Éponge de Menger: un cube (a), première (b), seconde (c) et troisième (d) itérations, schéma de règles associé (e) et mise en évidence de sommets encodés par les nœuds $n1$, $n7$, and $n16$.

Nœud $n1$:

```
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.3333333134651184);
res.addVect(p0);
Point3 p1 = Point3::middle(<0>_position(n0));
p1.scale(0.6666666865348816);
res.addVect(p1);
return res;
```

Nœud $n7$:

```
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.3333333134651184);
res.addVect(p0);
Point3 p2 = Point3::middle(<0,1>_position(n0));
p2.scale(0.6666666865348816);
res.addVect(p2);
return res;
```

Nœud $n16$:

```
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.3333333134651184);
res.addVect(p0);
Point3 p3 = Point3::middle(<0,1,2>_position(n0));
p3.scale(0.6666666865348816);
res.addVect(p3);
return res;
```

Lorsque nous appliquons les deux versions de l'opération - celle conçue manuellement et celle inférée à l'aide de l'approche proposée - sur un cube, elles fournissent le même le résultat : la première itération de l'éponge de Menger de la Figure 7b. Lorsque nous les appliquons sur d'autres objets, nous n'avons plus de garantie que le résultat sera le même, comme illustré sur la Figure 8. Soulignons que même si l'opération inférée ne correspond pas aux attentes de l'utilisateur, elle présente l'avantage d'être facilement disponible pour aider l'utilisateur à ajuster l'opération en cours de conception. Pour conclure, puisque l'éponge de Menger n'est formellement définie que dans le cas du cube, nous laissons le choix de la version préférée au goût esthétique du lecteur.

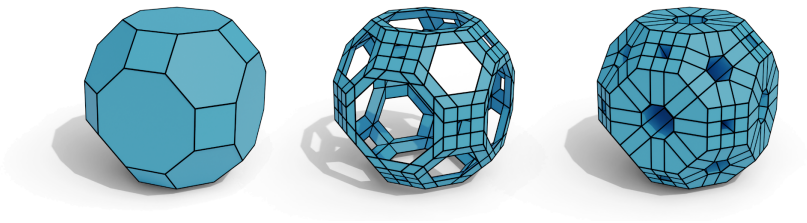


Figure 8: Illustration des deux calculs géométriques : (de gauche à droite) objet initial, résultat de la règle conçue manuellement, et résultat de l'opération inférée.

5 Conclusion

Nous avons décrit une approche générique pour l'inférence d'expressions géométriques afin d'être en mesure de générer automatiquement des opérations de modélisation géométrique prêtes à l'emploi. Pour nous guider dans la recherche de solutions, nous utilisons un langage dédié à base de règles de transformations de graphes. Nous avons introduit la notion de valeur d'intérêt (ici les barycentres des cellules topologiques) comme une valeur dont l'expression se calcule indépendamment de la topologie sous-jacente. Ces valeurs d'intérêt, couplées aux familles de fonctions (ici les combinaisons affines), permettent une représentation sous la forme d'un problème de satisfaction de contraintes que nous résolvons à l'aide de solveurs génériques. Par ailleurs, nous avons implanté notre méthode d'inférence géométrique au sein de l'outil JerboaStudio, comme extension de la plateforme Jerboa. Afin d'étoffer notre approche, nous avons l'intention d'étendre les familles des types de données et des valeurs d'intérêt associées.

References

[1] Agnès Arnould, Hakim Belhaouari, Thomas Bellet, Pascale Le Gall, and Romain Pascual. "Preserving consistency in geometric modeling with graph transformations". In: *Mathematical Structures in Computer Science* (Oct. 18, 2022). Publisher: Cambridge University Press, pp. 1–48. ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129522000226.

[2] Hakim Belhaouari, Agnès Arnould, Pascale Le Gall, and Thomas Bellet. "Jerboa: A Graph Transformation Library for Topology-Based Geometric Modeling". In: *Graph Transformation*. ICGT 2014. Ed. by Holger Giese and Barbara König. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 269–284. ISBN: 978-3-319-09108-2. DOI: 10.1007/978-3-319-09108-2_18.

[3] Guillaume Damiand and Pascal Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. CRC Press, Sept. 19, 2014. 407 pp. ISBN: 978-1-4822-0652-4.

[4] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Berlin Heidelberg: Springer-Verlag, 2006. ISBN: 978-3-540-31187-4. DOI: 10.1007/3-540-31188-2.

[5] Valentin Gauthier. "Développement d'un langage de programmation dédié à la modélisation géométrique à base topologique, application à la reconstruction de modèles géologiques 3D". These de doctorat. Poitiers, Jan. 17, 2019.

[6] Reiko Heckel and Gabriele Taentzer. *Graph Transformation for Software Engineers: With Applications to Model-Based Development and Domain-Specific Language Engineering*. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-43915-6. DOI: 10.1007/978-3-030-43916-3.

[7] Romain Pascual, Hakim Belhaouari, Agnès Arnould, and Pascale Le Gall. “Inferring topological operations on generalized maps: Application to subdivision schemes”. In: *Graphics and Visual Computing* 6 (May 14, 2022), p. 200049. ISSN: 2666-6294. DOI: 10.1016/j.gvc.2022.200049.

[8] Romain Pascual, Pascale Le Gall, Agnès Arnould, and Hakim Belhaouari. “Topological consistency preservation with graph transformation schemes”. In: *Science of Computer Programming* 214 (Feb. 1, 2022), p. 102728. ISSN: 0167-6423. DOI: 10.1016/j.scico.2021.102728.

On race detection in distributed systems using state models

Evgenii Vinarskii
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris, Palaiseau, France
vinarskii.evgenii@telecom-sudparis.eu

Abstract

This paper summarises the results of two papers devoted to race detection in distributed systems focusing on two complementary strategies: model checking [5] and model-based test generation [4]. We discuss how these approaches have been applied to the composition of a Software Defined Networking (SDN) controller and a switch resulting in the detection of races in the SDN framework. This study aims to provide the valuable insights for researchers and practitioners interested in race detection in distributed systems.

1 Introduction

Distributed systems, such as cloud computing and future networks, are becoming increasingly complex and can be sensitive to concurrency issues, such as races [3, 6]. The races can cause unexpected behaviour leading to incorrect responses, deadlocks, livelocks, and other issues [1]. Therefore, testing and verification against races in such systems are crucial [3]. In order to guarantee the absence of races, formal methods can be utilised.

An Input/Output automaton (IOA) is a formal model that is appropriate for describing possible races (Section 2). In [5], we study an Extended IOA (EIOA) as the specification of components of a distributed system and an LTL-formula for expressing a race-free property. For validation, we use the SPIN model checker [2]. If the race-free property is violated, SPIN returns a counterexample which represents the trace provoking the race in the distributed system (Section 3.1).

However, while the model checking approach is effective for detecting races in the specification, an implementation may not contain races, in this case we say that the implementation is race-free. To study races in the implementation, in [4], we propose a model-based testing approach against output races assuming that the specification is race-free. However, the model of an EIOA is not well-suited for testing, as there are no restrictions on how long a tester can wait for an output. To overcome this, as the specification we use a proper Timed Finite State Machine (TFSM) where inputs can be applied in a row without waiting for a produced output while every output is produced with a prescribed delay after a corresponding input has been applied. Correspondingly, races between different outputs can occur when outputs should be produced at the same time instance. In order to model races in implementations, we consider output delay mutants and study the properties of a timed transition tour for detecting output races in TFSM implementations (Section 3.2).

2 Background

In this section, we briefly introduce two main formal models which we use to describe the behaviour of distributed systems: an EIOA [5] and a TFSM [4].

EIOA $\mathcal{A}$ is a tuple (S, I, O, V, T, s_0) where S is a finite nonempty set of states with the designated initial state s_0; I (O) is a finite *input* (*output*) *alphabet*, $I \cap O = \emptyset$; V is a finite set of *context variables* with the set D_V of vectors of context variables' values; T is a set of transitions. A transition $(s, a, P, v_p, s') \in T$ means that being at state s and receiving (producing) symbol $a \in I \cup O$ when transition predicate P is true $\mathcal{A}$ moves to state s' and updates context variables with respect to context update transition function v_p. A *trace* $a_1 a_2 \dots a_n$ of $\mathcal{A}$ is a sequence of inputs and outputs corresponding to a sequence of transitions of $\mathcal{A}$. We say that $\mathcal{A}$ is

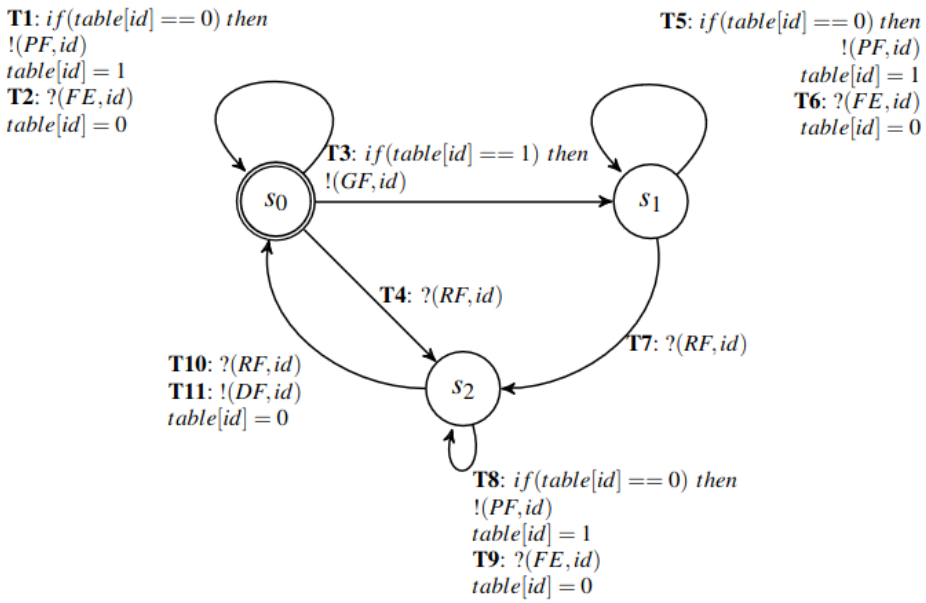


Figure 1: EIOA $\mathcal{C}$

output/output race-sensitive if there exist traces tr, tr' such that their input projections coincide, but output projections are different. Consider EIOA $\mathcal{C}$ in Fig. 1 that is used in [5] for describing the behaviour of an SDN controller with traces $tr = (PF,1)(GF,1)(PF,2)(FE,1)(FE,2)$ and $tr' = (PF,1)(GF,1)(PF,2)(FE,2)(FE,1)$. Since input projections of tr and tr' are equal to $(PF,1)(GF,1)(PF,2)$ and output projections are different $\mathcal{C}$ is output/output race-sensitive. In particular, outputs $(FE,1)$ and $(FE,2)$ compete to be produced.

TFSM $\mathcal{M}$ studied in [4] is a tuple $(S, I, O, G, h_S, D, s_0)$, where I (O) is a finite input (output) alphabet, $I \cap O = \emptyset$, S is a finite non-empty set of states, G is a finite non-empty set of integer intervals (timed guards), $h_S \subseteq (S \times I \times G \times O \times D \times S)$ is a set of transitions, D is a finite non-empty set of non-zero integer delays, s_0 is the *initial state*. A *transition* $(s, i, g, o, d, s') \in h_S$ means that if the machine receives input i after t time units being at state s, where $t \in g$, then the machine moves to state s' and produces output o after d time units. The next input can be applied before the machine has produced the output to the previous one. As an example, consider TFSM $C\&S$ shown in Fig. 2 that represents the composition of an SDN controller and a switch. Transition e_1 is enabled for timed input $(pf_1, 2)$ and $C\&S$ moves to state $F1$. Output fe_1 will be produced five time units later, in other words, there will be timed output $(fe_1, 7)$. Transition e_5, which moves $C\&S$ to state $F1\_F2$, is enabled for timed input $(pf_2, 4)$ since $(4-2) \in (1,3)$. Output fe_2 is produced two time units after applying input pf_2, i.e., there is timed output $(fe_2, 6)$, and the output response of $C\&S$ for the timed input sequence $(pf_1, 2)(pf_2, 4)$ is $fe_2 fe_1$, i.e., fe_2 is produced before fe_1 despite $C\&S$ receives pf_1 before pf_2. Thus, two different outputs can compete to be produced, and if they can be produced at the same time instance, we say that TFSM is *output/output race-sensitive*.

3 Race detection in distributed systems

In this section, we describe model checking and model-based test generation strategies for detecting races in distributed systems which were published in [5] and [4] respectively. We also present our preliminary experimental results in race detection in the SDN framework.

3.1 Model checking strategy for race detection

Given the composition of EIOAs describing the behaviour of a distributed system and an LTL-formula expressing the race-free behaviour, we use the SPIN model checker to verify whether the composition of EIOAs is race-sensitive, i.e., to check if there exist traces where the formula is violated. If the formula is violated, then SPIN produces a counterexample $\alpha = i_1 i_2 \dots i_n$ which provokes a race in the composition of EIOAs. That is, there are two traces tr, tr' such that their

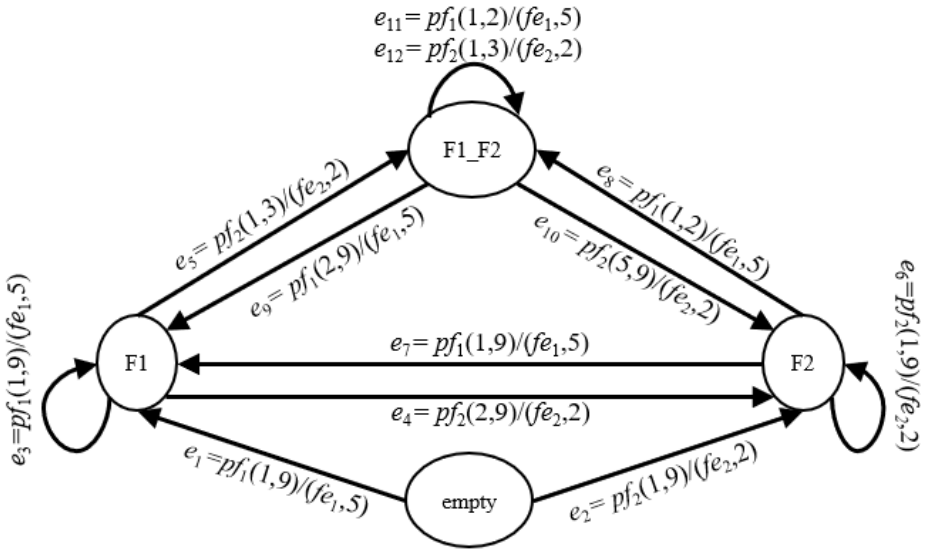


Figure 2: TFSM $C\&S$

input projections coincide while output projections are different, i.e., there are outputs that can compete to be produced.

As an experimental case study we consider the SDN framework consisting of an application, an ONOS SDN controller, an Open vSwitch and Openflow channels in Southbound interface. We derived the EIOAs for modelling their behaviour, for example Fig. 1 presents EIOA $\mathcal{C}$ for modelling the behaviour of the ONOS controller. For further model checking, namely races' detection, we described the EIOA in Promela language which is accepted by SPIN as an input. In order to detect a race, we considered the property $\varphi = \mathbf{G}((table[id] = 1) \rightarrow (table[id'] = 1))$ assuming that the rule with the id number is pushed before the rule with id' number and has a hard timeout value less or equal to that of id'. Formula φ guarantees that the id rule should be removed before the id' one. The counterexample produced by SPIN contained 10 requests. Following the algorithm introduced in [5] we augmented inputs with timestamps and observed an unexpected behaviour, i.e., the considered SDN framework is race-sensitive.

3.2 TFSM-based test generation strategy for race detection

In [4], we considered the race-free specification TFSM describing the behaviour of a distributed system. In order to model races, we introduced output delay mutants of the specification where the transition graph coincides with that of the specification but output delays of transitions can be different from those of the specification.

As a classical transition tour, i.e. a test suite which covers all transitions of an FSM, is known to detect many implementation faults, in [4], we focused on studying the properties of a timed transition tour, and its capabilities for detecting output races in TFSM implementations. We illustrated that (i) the choice of the timestamps significantly affects the fault coverage of a timed transition tour and (ii) unlike the classical one, the timed transition tour does not detect all output faults. As an application scenario in [4], we considered Software Defined Networking systems and related components. Experimental results show that output races in the SDN components can be detected by a timed transition tour. Consider the specification TFSM $C\&S$ in Fig. 2 that describes the behaviour of the composition of ONOS controller and switch and timed transition tour $ttt = \{(pf_1, 2)(pf_2, 4)(pf_1, 5.5)(pf_2, 7.5)\ (pf_1, 10)(pf_1, 12), (pf_2, 2)(pf_2, 8)(pf_2, 11)(pf_1, 13)\ (pf_1, 14.5)(pf_2, 20)\}$.
As well as in Section 3.1 we executed ttt and observed an unexpected behaviour, i.e., the SDN framework is race-sensitive.

4 Conclusion

This paper summarises results on detecting races in distributed systems presented in papers [4, 5]. Our current work consists of deriving a test suite with the guaranteed fault coverage against races.

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[2] Gerard Holzmann. *The Spin model checker: primer and reference manual*. Addison-Wesley Professional, 2003.

[3] Razvan Raducu, Ricardo J. Rodríguez, and Pedro Álvarez. Defense and attack techniques against file-based TOCTOU vulnerabilities: A systematic review. *IEEE Access*, 10:21742–21758, 2022.

[4] Evgenii Vinarskii, Natalia Kushik, Jorge López, Nina Yevtushenko, and Djamal Zeghlache. Timed transition tour for race detection in distributed systems. In *18th Int. Conf. on Evaluation of Novel Approaches to Software Engineering*, pages 613–620, 2023.

[5] Evgenii Vinarskii, Jorge López, Natalia Kushik, Nina Yevtushenko, and Djamal Zeghlache. A model checking based approach for detecting SDN races. In *31st IFIP WG 6.1 Int. Conf. on Testing Software and Systems*, pages 194–211, 2019.

[6] Cheng Wen, Mengda He, Bohao Wu, Zhiwu Xu, and Shengchao Qin. Controlled concurrency testing via periodical scheduling. In *IEEE/ACM 44th Int. Conf. on Software Engineering*, pages 474–486, 2022.

Une logique de séparation de haut niveau pour l'espace de tas en présence d'un glaneur de cellule

Alexandre Moine[1], Arthur Charguéraud[2], and François Pottier[1]

[1]Inria, Paris, France
[2]Inria & Université de Strasbourg, CNRS, ICube, Strasbourg, France

Ce texte est un résumé de notre article « A High-Level Separation Logic for Heap Space under Garbage Collection » présenté à POPL en 2023 [5].

La vérification d'un programme porte le plus souvent sur sa *sûreté* et sa *correction fonctionnelle* : l'objectif est de montrer que le programme ne se retrouve jamais dans une configuration bloquée et qu'il calcule un résultat correct. Dans le domaine de la vérification déductive de programme [2], un programme est généralement vérifié à l'aide d'une logique de programme, c'est-à-dire un ensemble de règles de déduction dont la correction a été établie une fois pour toutes. Les logiques de séparation [6, 1, 3] sont de telles logiques de programme : elles permettent de raisonner de manière compositionnelle en présence de fonctionnalités avancées comme l'allocation dynamique de mémoire et l'état mutable.

Au-delà de la sûreté et de la correction fonctionnelle, il est souhaitable d'établir des bornes sur les ressources consommées par un programme. Par exemple, un programme prenant trop de temps à s'exécuter peut arrêter de répondre ou un programme utilisant trop d'espace de tas peut rendre le système instable.

Notre travail s'intéresse à borner l'espace de tas en présence d'un *glaneur de cellule* (GC, « garbage collector » en anglais). Dans ce cadre, il n'y a pas d'instruction explicite de désallocation. En effet, le GC peut s'exécuter à n'importe quel moment pour désallouer des blocs mémoires devenus inaccessibles.

Un travail précédent [4] propose une logique de séparation avec des *crédits-espace*, une ressource fantôme pour compter l'espace libre, et des assertions « pointed-by » pour garder trace des prédécesseurs des blocs mémoires. Dans cette logique, l'allocation d'un bloc de taille n consomme n crédits-espace. Symétriquement, lorsque l'utilisateur est capable de montrer que ce bloc n'a plus de prédécesseurs, c'est qu'il est inaccessible et la logique produit n crédits-espace. Cependant, le langage-cible de cette logique est de bas niveau, proche de l'assembleur.

Notre travail étend cette logique à un langage de haut niveau, proche d'OCaml. Une question centrale est d'identifier les *racines* choisies par le GC, c'est-à-dire l'ensemble d'adresses servant à calculer les blocs accessibles. Nous proposons une nouvelle assertion (appelée $\mathit{Stackable}$) de logique de séparation pour suivre l'évolution des racines au cours de l'exécution du programme. De plus, nous proposons des règles de raisonnement pour les *fermetures*, les structures concrètes allouées sur le tas qui implémentent le concept de fonction de première classe.

Nous illustrons notre logique sur plusieurs exemples, allant des fonctions récursives sur les listes chaînées, à des objets implémentés à l'aide de fermetures avec état interne mutable. Tous nos résultats sont prouvés en Coq au-dessus du système de logique de séparation Iris [3].

Références

[1] Stephen Brookes and Peter W. O'Hearn. Concurrent separation logic. *SIGLOG News*, 3(3) :47–65, 2016.

[2] Jean-Christophe Filliâtre. Deductive software verification. *Software Tools for Technology Transfer*, 13(5) :397–403, 2011.

[3] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up : A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming*, 28 :e20, 2018.

[4] Jean-Marie Madiot and François Pottier. A separation logic for heap space under garbage collection. *Proceedings of the ACM on Programming Languages*, 6(POPL), January 2022.

[5] Alexandre Moine, Arthur Charguéraud, and François Pottier. A high-level separation logic for heap space under garbage collection. *Proc. ACM Program. Lang.*, 7(POPL), jan 2023.

[6] John C. Reynolds. Separation logic : A logic for shared mutable data structures. In *Logic in Computer Science (LICS)*, pages 55–74, 2002.

Approche Formelle Dirigée par les Modèles pour la Collaboration de DSLs

Salim Chehida, Akram Idani, Mario Cortes-Cornax, German Vega

Univ. Grenoble Alpes, Grenoble INP, CNRS, LIG
F-38000 Grenoble France

{prenom.nom}@univ-grenoble-alpes.fr

Résumé

Ce travail présente une extension de la plate-forme **Meeduse** permettant de faire collaborer des modèles issus de différents langages dédiés domaines (ou DSLs). **Meeduse** est un atelier de conception formelle de DSLs qui repose sur la méthode B. Il permet de spécifier en B la syntaxe abstraite du langage ainsi que sa sémantique dynamique. Cependant, l'outil se limite à un seul langage à la fois. Aussi, son applicabilité à des contextes réalistes est-elle restreinte car un système informatique est souvent conçu au travers d'une panoplie de modèles. Dans ce travail nous proposons de l'étendre en vue de couvrir la composition et la coordination de plusieurs DSLs. Ces aspects sont d'abord capturés via un modèle BPMN (Business Process Model and Notation) et ensuite traduits dans CSP (Communication Sequential Process) afin de définir un cadre rigoureux favorisant l'animation et la vérification. Notre approche a été appliquée avec succès sur une étude de cas réel fournie par RTE, le gestionnaire du réseau de transport d'électricité français.

Ce document est un résumé long de l'article suivant :

> Salim Chehida, Akram Idani, Mario Cortes-Cornax and German Vega. *A Formal MDE Framework for inter-DSL Collaboration*. In 25$^{\text{th}}$ International Conference on Coordination Models and Languages, 2023 (Accepted).

1 Introduction

Un langage dédié domaine (appelé DSL ou Domain-Specific Language) permet de spécifier des modèles adaptés à un domaine d'application particulier ou à un aspect spécifique du système. Dans de nombreux cas, un ensemble de modèles provenant de DSLs variés et indépendants sont combinés pour construire les différentes facettes d'un système. Ce faisant, il est nécessaire d'établir des liens explicites entre ces modèles et gérer leur inter-dépendance. Si la composition de la sémantique statique de ces DSLs a déjà été abordée par plusieurs travaux (*e.g.* [5, 3]), la coordination de leurs sémantiques d'exécution reste un défi qui a été très peu étudié.

Dans ce travail, nous abordons cette problématique en proposant une approche qui permet de définir comment les DSLs collaborent entre eux. Notre approche est supportée par la plate-forme **Meeduse** [4] donnant lieu à un cadre formel utile à la vérification de cette collaboration. La figure 1 est une illustration de l'architecture mise en oeuvre dans le cadre de ce travail. Celle-ci est divisée en deux principaux blocs (A et B) qui seront décrits dans la section 2. Le bloc A applique **Meeduse** pour la conception de plusieurs DSLs. On considère dans ce schéma que nous disposons de deux DSLs : `DSL_X et DSL_Y`. Le bloc B schématise notre contribution à la plate-forme **Meeduse** pour la modélisation, l'animation et la vérification de la collaboration entre ces DSLs. L'extension proposée est constituée de deux éléments majeurs : un méta-modèle dédié à la composition des sémantiques statiques des DSLs, et un modèle BPMN dédié à la coordination de leurs exécutions.

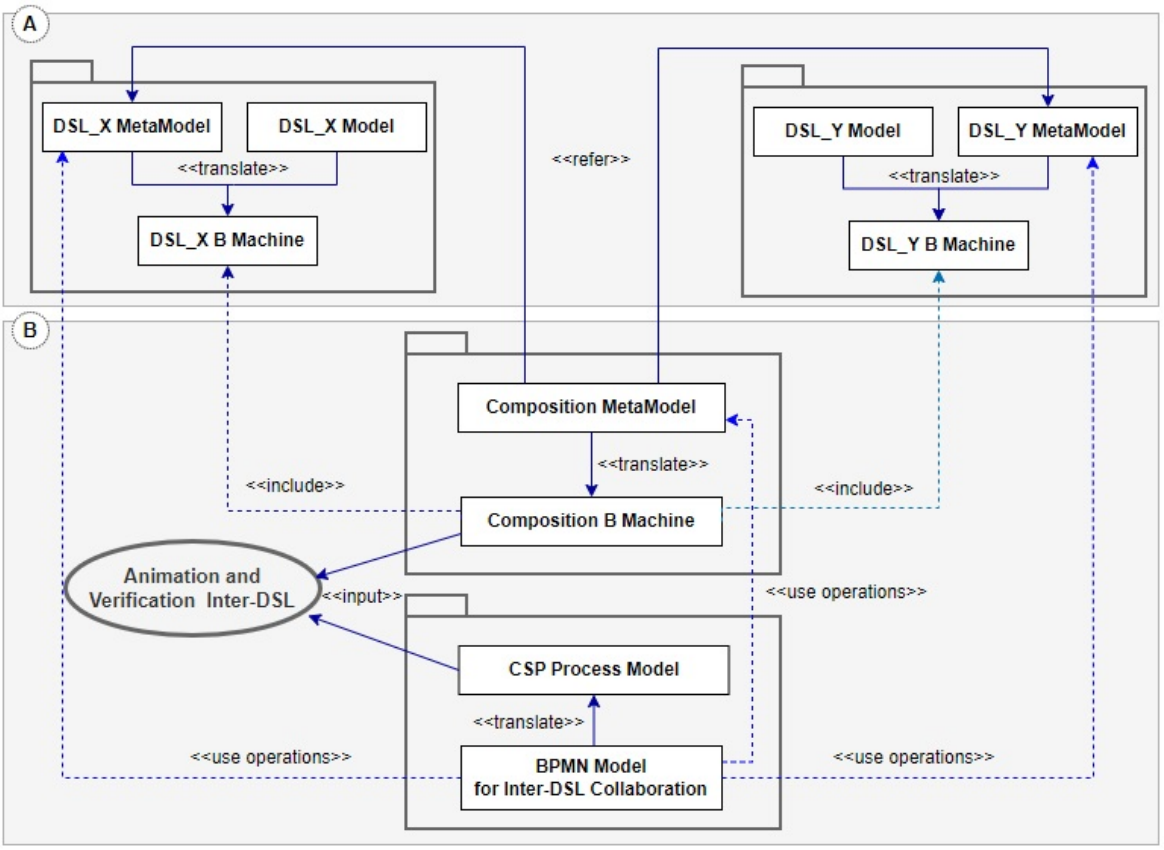


Figure 1: Extension de **Meeduse** pour la collaboration de DSLs

2 Approche

2.1 Modélisation et formalisation de DSLs

Dans le domaine de l'IDM, la sémantique statique d'un DSL est souvent définie au moyen d'un méta-modèle. Ce dernier représente les concepts du DSL par des méta-classes caractérisées par un ensemble d'attributs et d'opérations, reliés par un ensemble d'associations. Ayant un méta-modèle, la plate-forme **Meeduse** produit une machine B qui définit les caractéristiques structurelles du DSL par des ensembles abstraits, des variables et des invariants de typage. Dans cette étape, l'outil applique une transformation UML-vers-B classique, étant donné qu'un méta-modèle est au final un diagramme de classes. La machine B résultant de cette traduction permet au concepteur d'introduire les propriétés invariantes que le langage doit respecter ; et ce, qu'il soit exécutable ou non.

Quant à la sémantique dynamique du DSL, qui décrit les actions réalisées par le langage lors de son exécution, elle peut être définie par des opérations B au sein de cette même machine. Ce cadre formel permet de travailler sur la correction du langage au moyen d'outils de preuve tel que l'AtelierB. L'avantage de **Meeduse** est qu'il va au delà de la traduction en B du méta-modèle. En effet, il permet d'animer une instance du méta-modèle en faisant tourner ProB [6] de manière complètement transparente. L'exécution du DSL est ainsi réalisée via l'animation des opérations B de sa sémantique dynamique. Toutefois, cette approche permet d'exécuter un seul DSL, ou plusieurs DSLs indépendamment les uns des autres. Il n'est donc pas possible de voir l'impact que l'exécution d'un modèle pourrait avoir sur les autres modèles. En vue d'aborder cette limitation il est nécessaire d'expliciter les liens entre les sémantiques (statique et dynamique) des différents DSLs.

2.2 Modélisation de la collaboration entre DSLs

Nous proposons de spécifier la collaboration entre les DSLs par un méta-modèle de composition et un diagramme BPMN. Le premier établit les liens entre les sémantiques statiques, et le deuxième permet l'orchestration des sémantiques dynamiques. Ces deux modèles sont ensuite traduits en spécifications formelles. Le méta-modèle de composition est traduit en B en bénéficiant des aspects compositionnels de la méthode B ; et le modèle BPMN est traduit en CSP, favorisant ainsi à la fois l'animation de plusieurs DSLs et la vérification de la collaboration.

2.2.1 Composition

Le méta-modèle de composition fait référence aux méta-modèles des DSLs qu'on souhaite faire collaborer. Il est représenté par une méta-classe racine reliée par des associations de composition aux méta-classes racines de chaque DSL à coordonner. À ce méta-modèle, le concepteur peut ajouter de nouvelles méta-classes avec de nouveaux attributs, opérations (actions de composition) et références pour répondre aux besoins de la collaboration entre les DSLs en entrée. À partir de ce méta-modèle, **`Meeduse`** génère la machine B de composition. Celle-ci inclut les spécifications B obtenues à partir des différents DSLs, ce qui lui donne accès à leurs opérations. Dans la partie dynamique de la machine de composition, nous intégrons les opérations B définissant les actions impliquées dans le processus de collaboration. Il est également possible d'introduire des propriétés invariantes indiquant les conditions que la composition doit respecter.

2.2.2 Coordination

Nous exprimons la coordination des actions des DSLs en utilisant la norme BPMN (Business Process Model and Notation). Dans notre approche, nous utilisons la notion de pool BPMN pour regrouper les opérations de chaque DSL, y compris les opérations du méta-modèle de composition. Nous représentons une action atomique spécifiant une opération du méta-modèle par une tâche BPMN et nous utilisons un sous-processus pour représenter un regroupement de tâches. Les flux de séquences BPMN sont utilisés pour représenter la séquence d'actions dans le contexte d'un DSL (à l'intérieur du pool), tandis que les flux de messages sont utilisés pour représenter la communication inter-pool. Les passerelles (exclusives ou parallèles) modélisent le flux de contrôle dans chaque DSL.

Pour permettre l'animation et la vérification de la collaboration de DSLs, nous traduisons les diagrammes BPMN en CSP. En CSP, un processus représente une entité indépendante qui exécute une séquence d'évènements, ce qui est similaire à la notion de pool en BPMN. Nous transformons les pools en processus CSP indépendants, puis nous produisons un processus principal pour les synchroniser en étant guidés par les messages échangés par les pools. La communication entre les processus dans CSP est assurée par des canaux, qui peuvent ou non transmettre des flux de données. Nous utilisons donc cette notion pour représenter les messages échangés entre les pools ainsi que les tâches atomiques du modèle BPMN. La formalisation en B du méta-modèle de composition et en CSP du modèle BPMN, alimente l'outil ProB [6] (intégré dans **`Meeduse`**) pour l'animation et la vérification de la collaboration. Pour ce faire, nous appliquons l'approche CSP||B, telle que présentée dans [1].

3 Application

Nous avons appliqué notre approche à une étude de cas sur le réseau electrique intelligent fournie par RTE, la société de transmission d'énergie en France. L'étude de cas implique deux DSLs : le premier, appelé CM-DSL (Configuration Management DSL), se concentre sur la gestion des configurations système attribuant à un ensemble d'applications différentes infrastructures. Le deuxième, nommé SRA-DSL (Security Risk Assessment DSL), est dédié à l'évaluation des risques de sécurité. Ces deux DSLs sont indépendants l'un de l'autre, et permettent, chacun, de se focaliser sur une préoccupation particulière du système. Les faire collaborer ensemble permet de gérer les configurations système tout en évaluant les risques qui en découlent. Cela vise à identifier les configurations

système les moins risquées et à établir les défenses nécessaires quand la configuration n'atteint pas un niveau de sécurité satisfaisant.

Dans [2], nous décrivons les différentes étapes de notre approche et son application à notre étude de cas. Nous fournissons également les différents artefacts de cette étude de cas. La partie modélisation comprend les méta-modèles CM-DSL et SRA-DSL, le méta-modèle de leur composition, les modèles qui définissent les instances des méta-modèles, et les diagrammes BPMN qui décrivent la collaboration entre les DSLs. La partie spécification formelle comprend les machines B des DSLs et de leur composition ainsi que le modèle CSP issu des diagrammes BPMN.

Nous avons testé plusieurs exécutions de notre modèle de collaboration basé sur notre étude de cas. Cela a conduit à la création de plusieurs configurations sécurisées tout en identifiant les menaces et les défenses sous-jacentes. En ce qui concerne la vérification, nos spécifications formelles ont été prouvées, ce qui garantit la correction des modèles B sur lesquels nous avons construit le processus de collaboration.

4 Conclusion

Dans ce résumé, nous avons décrit une approche qui combine les techniques de composition et de coordination. Nous construisons d'abord un méta-modèle de composition qui fait référence aux méta-modèles issus de divers DSLs sans altérer ni leurs structures ni leurs comportements. Le diagramme BPMN est utilisé pour exprimer le processus de coordination entre les DSLs. Les méta-modèles des DSLs et le modèle BPMN sont ensuite transformés en une spécification formelle utile afin d'animer, vérifier et valider la collaboration de modèles issus de ces DSLs.

References

[1] Butler, M., Leuschel, M.: Combining CSP and B for Specification and Property Verification. In: Fitzgerald, J., Hayes, I.J., Tarlecki, A. (eds.) FM 2005: Formal Methods. pp. 221–236. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[2] Chehida, S., Idani, A., Cortes-Cornax, M., Vega, G.: GitHub Artifacts, https://github.com/SalimChehida/Inter-DSL-Collaboration

[3] Emerson, M., Sztipanovits, J.: Techniques for metamodel composition. OOPSLA - 6th Workshop on Domain Specific Modeling (01 2006)

[4] Idani, A.: Meeduse: A Tool to Build and Run Proved DSLs. In: Dongol, B., Troubitsyna, E. (eds.) 16th International Conference on Integrated Formal Methods, IFM 2020. LNCS, vol. 12546, pp. 349–367. Springer (2020), https://doi.org/10.1007/978-3-030-63461-2_19

[5] Jouault, F., Vanhooff, B., Bruneliere, H., Doux, G., Berbers, Y., Bezivin, J.: Inter-dsl coordination support by combining megamodeling and model weaving. In: Proceedings of the 2010 ACM Symposium on Applied Computing. p. 2011–2018. SAC '10, Association for Computing Machinery, New York, NY, USA (2010)

[6] Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. International Journal on Software Tools for Technology Transfer **10**, 185–203 (2008)

Débogage Multivers de Modèles UML

Matthias Pasquier
matthias.pasquier@ertosgener.com

Ciprian Teodorov
ciprian.teodorov@ensta-bretagne.fr

Frédéric Jouault
frederic.jouault@eseo.fr

Matthias Brun
matthias.brun@eseo.fr

Loïc Lagadec
loic.lagadec@ensta-bretagne.fr

Résumé

Pour faciliter la phase de spécification, nous présentons un débogueur multivers pour les machines à état UML ainsi que deux extensions à cette approche : les breakpoints temporels permettent l'utilisation de multiples logiques temporelles pour explorer le modèle. La réduction permet un certain contrôle sur l'espace d'état exploré pour permettre le passage à l'échelle.

1 Introduction

Le débogage est une étape importante dans le cycle de développement logiciel, et peut être bénéfique des les phases de spécification. Le débogage multivers [1] est une extension des méthodes de débogage classiques, permettant l'exploration des systèmes non-déterministes. Cet article présente l'implémentation d'un tel débogueur pour l'environnement AnimUML [2], dédié aux machines à état UML. Cette approche est ensuite étendue par deux nouvelles contributions. D'une part, l'amélioration de l'expressivité des breakpoints comme introduit dans [3], les automates AnimUML peuvent également être utilisés en tant qu'automates de propriétés. D'autre part, dans [4], nous proposons une solution pour le passage à l'échelle du débogage multivers, en permettant à l'utilisateur de définir des réductions limitant la façon dont le système peut être exploré lors de la recherche de breakpoints.

2 Le débogage multivers de modèles UML

Les langages de spécification, tels que les langages de machine à état, présentent des caractéristique non-déterministes, qui présentent un challenge pour le débogage, en partie adressé par le débogage multivers [1]. Nous présentons ici notre implémentation de débogueur utilisée sur des machines à état UML, ainsi que les extensions que nous avons apportées à cette technique pour faciliter son utilisation.

Aperçu de l'approche La figure 1 présente l'approche en commençant par les interfaces dédiées à l'utilisateur en haut. Pour lancer le processus de débogage, l'utilisateur fournit une spécification au débogueur qui instancie la sémantique appropriée du langage sujet.

Le débogueur propose alors la syntaxe des actions de debug, contenant :

1) `step` pour passer à la configuration suivante, *2)* `jump` pour revenir à une configuration explorée précédemment, *3)* `select` pour résoudre les cas de multiples configurations parallèles, et *4)* `run_to_breakpoint` pour chercher à atteindre un point particulier dans l'exécution du système.

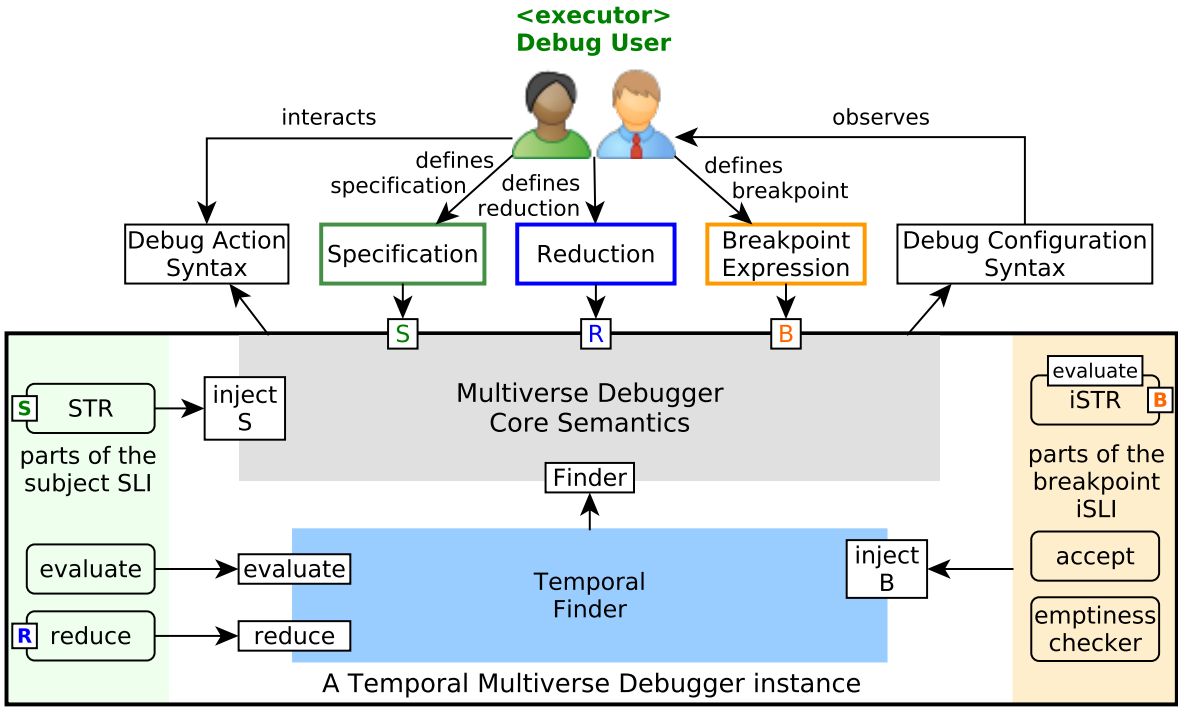


FIGURE 1 – Aperçu du Debogueur Multivers Réduit avec breakpoints temporels

Au cours d'une session, l'utilisateur peut inspecter la configuration de débogage actuelle, qui contient l'ensemble des traces d'exécution déjà explorées ainsi que la configuration de la mémoire à chaque point d'exécution.

Pour l'utilisation de l'action `run_to_breakpoint`, deux autres interfaces sont disponibles pour l'utilisateur. La première permet de définir une expression de point d'arrêt conditionnelle. La seconde permet de définir une réduction. L'objectif et l'utilisation de ces deux interfaces seront l'objet des deux prochaines sections.

La seconde partie de la figure résume les grands principes de l'architecture de notre débogueur. Les deux cotés de l'instance du débogueur représentent le langage sujet à gauche et le langage d'expression de breakpoint à droite. Ces deux langages sont indépendants l'un par rapport à l'autre, et détachés du fonctionnement du débogueur au travers d'une interface d'exécution commune, la Semantic Language Interface ou SLI. Elle regroupe le pilotage de l'exécution du système, ainsi que des interfaces pour obtenir des informations sur ses configurations [4].

Pour obtenir une instance de débogage multivers temporel, l'utilisateur de l'outil doit fournir à la fois la sémantique du langage concerné et la sémantique des expressions de point d'arrêt, créant ainsi une instance spécifique de débogueur adaptée à ces langages. Le composant Finder, en charge de la recherche de breakpoint, prend en entrée les éléments nécessaires à une composition synchrone de l'exécution du sujet et du breakpoint, puis effectue une phase de model-checking afin de tenter d'atteindre une configuration satisfaisante.

Des breakpoints temporels pour le multivers Les breakpoints sont habituellement exprimées sur une unique configuration, c'est-à-dire un point précis dans le fil d'exécution du système. La littérature présente de nombreux types de breakpoints additionnels, soulignant le besoin de logiques plus expressives, afin d'exprimer des conditions sur le reste de l'exécution (passée ou future) du système en dehors de la configuration actuellement présente dans le débogueur. Pour permettre une telle expressivité dans un environnement multivers, nous définissons le concept de langage de breakpoint, contenant une sémantique ainsi que les règles à suivre pour l'exploration de la composition avec le système.

Pour permettre à l'utilisateur d'utiliser la logique la plus adaptée, notre architecture découple

le débogueur et le langage de breakpoint. Celle-ci nous permet d'écrire des conditions d'arrêt plus complexe. Les conditions sur le pas d'exécution menant à la configuration actuelle nous permettent de simuler des watchpoints, ou d'arrêter l'exécution sur une action spécifique. Autrement, un langage basé sur les NFA (Automates Finis Non-déterministes) permet de poser des conditions sur le chemin d'exécution ayant mené à la configuration actuelle. Une condition doit être préalablement validée, un point doit avoir été visité un certain nombre de fois... Finalement, l'utilisation d'automates de Büchi permet de s'arrêter sur des conditions futures, permettant par exemple de décrire une situation de livelock.

Réduire le multivers Les breakpoints temporels, inclus dans l'environnement du débug multivers permettent une exploration plus facile des programmes non-déterministes. Cependant, cette méthode souffre de problèmes de passage à l'échelle lors de cette recherche. Ce problème est dû à l'exploration exhaustive de l'espace d'état potentiellement infini du système. Pour résoudre ce problème, nous utilisons un débogueur multivers réduit, permettant à l'utilisateur de définir des réductions pour atteindre un breakpoint. Les réductions sont définies comme des fonctions prenant en entrée une configuration du système en cours d'exploration, et retournant une valeur arbitraire, qui sera utilisée pour vérifier l'égalité entre deux configurations. Durant l'exploration, une configuration sera considérée comme déjà visitée si sa valeurs réduite est identique à une configuration explorée précédemment. Cela nous permet de limiter et d'orienter l'exploration, par exemple en supprimant des variables, en ajoutant des prédicats, ou en passant la configuration dans une fonction de hachage. Cela nous permet de répliquer de nombreuses approches d'abstraction présentes dans la littérature du model-checking, permettant effectivement de couper à la volée une partie des branches qui ne seront pas explorées dans l'arbre des possibilités.

Application à UML Pour valider notre approche, nous avons mis en place un débogueur multivers pour l'environnement de modélisation AnimUML. Pour cela, nous avons adapté le moteur AnimUML permettant le pilotage de machine à état UML par le biais d'une interface SLI. Concernant les langages de breakpoints, nous proposons 3 formalismes différents : les expressions régulières, les machines à état, et les automates de Büchi. Par l'utilisation d'une interface commune, les modèles AnimUML peuvent également être utilisés comme breakpoints. Une formalisation [1] de notre approche en Lean est également disponible afin d'offrir une preuve de sa bonne construction.

3 Conclusion

Cet article présente notre approche permettant l'obtention d'un débogueur multivers instancié sur des machines à état UML. Les breakpoints temporels permettent une recherche plus précise sur les conditions de breakpoints, et les réductions rendent l'ensemble de l'approche adaptable à grande échelle.

Références

[1] R. G. Singh, C. T. Lopez, S. Marr, E. G. Boix, and C. Scholliers, "Multiverse Debugging : Non-Deterministic Debugging for Non-Deterministic Programs (Artifact)," *Dagstuhl Artifacts Series*, vol. 5, no. 2, pp. 4 :1–4 :3, 2019.

[2] F. Jouault, V. Besnard, T. L. Calvar, C. Teodorov, M. Brun, and J. Delatour, "Designing, animating, and verifying partial uml models," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '20, (New York, NY, USA), p. 211–217, Association for Computing Machinery, 2020.

1. https ://github.com/teodorov/temporal-multiverse-debugging

[3] V. Besnard, C. Teodorov, F. Jouault, M. Brun, and P. Dhaussy, "Unified verification and monitoring of executable uml specifications," *Software and Systems Modeling*, vol. 20, no. 6, pp. 1825–1855, 2021.

[4] M. Pasquier, C. Teodorov, F. Jouault, M. Brun, L. L. Roux, and L. Lagadec, "Practical multiverse debugging through user-defined reductions : Application to uml models," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, MODELS '22, (New York, NY, USA), p. 87–97, Association for Computing Machinery, 2022.

Vérification de modèles relationnels et temporels avec Pardinus*

Nuno Macedo[1], Julien Brunel[2], David Chemouil[2], and Alcino Cunha[3]

[1]INESC TEC, FEUP, Porto
[2]ONERA DTIS, Toulouse, France , Université de Toulouse
[3]INESC TEC , University of Minho, Braga

Abstract

Nous résumons ici un article paru dans le *Journal of Automated Reasoning* [5]. Celui-ci présente Pardinus, une extension du *model finder* relationnel Kodkod [9] au moyen de la logique temporelle linéaire (avec opérateurs du passé). Pardinus inclut un moteur de *bounded model-checking* basé sur SAT ainsi qu'un moteur de *model-checking* commplet basé sur SMV, les deux permettant d'itérer sur les instances (ou contre-exemples) d'une spécification. Il offre aussi une stratégie d'analyse « décomposée » parallèle qui améliore l'efficacité des deux moteurs d'analyse.

1 Introduction

Les *model finders* de haut niveau deviennent de plus en plus utiles pour le génie logiciel. La possibilité de spécifier les propriétés d'un système dans une logique expressive, puis trouver automatiquement des solutions qui satisfont ces propriétés (des modèles, au sens de la logique) est utile dans de nombreux applications, allant de la validation précoce de la conception d'un système à la génération de cas de test. Kodkod [9] est un *model finder* proposant une gamme de fonctionnalités qui le rendent très populaire :

- Les problèmes sont décrits en utilisant le concept unique de relation (d'arité arbitraire), simplifiant considérablement la syntaxe et la sémantique du langage.
- Les contraintes sont exprimées en logique relationnelle (logique du premier ordre enrichie de l'algèbre relationnelle, incluant un opérateur de fermeture transitive) permettant un style de spécification succinct et lisible à la fois.

*Travaux financés par le *European Regional Development Fund (ERDF)* à travers *l'Operational Programme for Competitiveness and Internationalisation (COMPETE2020)* et par des fonds nationaux à travers la *Fundação para a Ciência e a Tecnologia (FCT)* dans le projet POCI-01-0145-FEDER-016826 et les projets FORMEDICIS (ANR-16-CE25-0007) et CONCORDE (AID, Ministère de la Défense, 2019650090004707501).

- Il permet à l'utilisateur d'itérer sur des instances alternatives du problème, tout en mettant en place un mécanisme de cassage de symétrie (pour éviter la génération de instances équivalentes), ce qui le rend utile pour l'exploration de scénarios.
- Des instances partielles peuvent être fournies *a priori*, comme bornes inférieure et supérieure des relations, permettant l'application de Kodkod aux tâches de résolution de configuration, où le but est de trouver une instanciation complète d'une description partielle d'un système.

Kodkod est implémenté en tant qu'API Java et est conçu pour être un plugin qui peut facilement être incorporé en tant que *backend* d'un autre outil. Son application la plus connue est l'analyse des spécifications d'Alloy. Alloy [4] est un langage qui partage certaines caractéristiques de Kodkod —utilisation de la logique relationnelle notamment— mais qui prend également en charge un système de types avec héritage pour simplifier davantage la description d'un système. Malgré son utilité et sa popularité, Kodkod ne peut être directement appliqué qu'à l'analyse de modèles *structurels*. L'analyse de modèles comportementaux est possible, mais fastidieuse et propice aux erreurs.

Le présent travail introduit le *model finder* Pardinus, une extension de Kodkod qui répond à cette limitation. Il permet la déclaration de relations mutables et la spécification des propriétés en logique relationnelle *temporelle*, une extension de la logique relationnelle au moyen de la logique temporelle linéaire avec opérateurs du passé (PLTL). Les problèmes peuvent être analysés par deux moteurs d'analyse : le premier traduit les problèmes Pardinus en simples problèmes Kodkod, en implémentant une procédure qui revient essentiellement au *bounded model checking* basé sur des techniques SAT [1] ; le second déplie le domaine du premier ordre et réduit la vérification de satisfiabilité au *model checking* de PLTL sur un modèle universel de système (celui qui permet à tous comportements possibles) [7], en utilisant les outils SMV [3]. L'application principale de Pardinus est l'analyse des spécifications Alloy dans sa version 6. Cette nouvelle version d'Alloy ajoute la prise en charge des relations mutables et de la logique relationnelle temporelle (une extension précédemment connue sous le nom d'Electrum [6, 2]). Pardinus est également utilisé comme *backend* dans Forge [8], un système pour prototyper des outils de méthodes formelles. Un exemple de spécification Pardinus est présenté dans la figure 1. Nous résumons dans la suite quelques aspects saillants de Pardinus présentés dans [5].

2 Itération sur les solutions

Une fois qu'une solution (ou un contre-exemple) est calculée par Pardinus, un mécanisme permet une itération sur l'ensemble des solutions (ou contre-exemples). L'approche de Kodkod (à savoir renvoyer n'importe quel chemin différent) échouerait souvent à intégrer les attentes des utilisateurs lors de l'exploration de chemins alternatifs. Dans notre expérience, l'exploration de scénarios est souvent réalisée en étapes distinctes. Par exemple, l'utilisateur peut d'abord explorer différentes configurations, chacune encadrant le contexte dans lequel un chemin peut évoluer, puis explorer des chemins alternatifs pour une configuration sélectionnée, essayer de trouver un scénario d'évolution intéressant. Ainsi, Pardinus implémente différentes opérations de navigation qui modifient différents aspects du chemin. Pour être efficace, ces opérations sont

```
1  {I0,I1,I2,I3,P0,P1,P2,P3}
2
3      Id      :1 {(I0),(I1),(I2),(I3)} {(I0),(I1),(I2),(I3)}
4      next    :2 {(I0,I1),..,(I2,I3)} {(I0,I1),..,(I2,I3)}
5      Process :1 {} {(P0),(P1),(P2),(P3)}
6      id      :2 {} {(P0,I0),(P0,I1),(P0,I2),(P0,I3),..,
7                    (P3,I0),(P3,I1),(P3,I2),(P3,I3)}
8      succ    :2 {} {(P0,P0),(P0,P1),(P0,P2),(P0,P3),..,
9                    (P3,P0),(P3,P1),(P3,P2),(P3,P3)}
10 var outbox  :2 {} {(P0,I0),(P0,I1),(P0,I2),(P0,I3),..,
11                   (P3,I0),(P3,I1),(P3,I2),(P3,I3)}
12 var Elected :1 {} {(P0),(P1),(P2),(P3)}
13
14 id in Process → Id and
15 all p : Process | one p.id and
16 all i : Id | lone id.i and
17 succ in Process → Process and
18 all p : Process | one p.succ and
19 all p : Process | Process in p.^succ and
20
21 outbox = id and
22 always some p : Process, i : (succ.p).outbox |
23   outbox' = outbox − succ.p → i + p → (i − ^next.(p.id)) and
24
25 always Elected = {p : Process |
26     once (p.id in p.outbox and before not (p.id in p.outbox))}
```

Figure 1: Protocole d'élection d'un *leader* en Pardinus

directement implémentées au niveau du solveur, et intégrent aussi un mécanisme de cassage de symétries.

3 Décomposition parallèle

Les « configurations », déterminées par les relations immuables, sont initialement arbitraires, mais restent constantes au fur et à mesure que le système évolue. Cela permet une analyse décomposée de problèmes, en résolvant d'abord les configurations et ensuite, pour chaque configuration, en analysant les comportements possibles. L'évaluation montre que dans certains contextes, cette décomposition peut apporter des avantages substantiels en termes de performances. L'analyse « décomposée » se prête également à la parallélisation car différentes configurations peuvent être résolues indépendamment dans différents cœurs. De plus, puisque généralement les valeurs des relations mutables dépendent de celles des relations immuables, si ces dépendances étaient explicites, les configurations pourraient être utilisées en tant qu'instances partielles pour l'étape suivante, accélérant encore l'analyse. Dans cette perspective, Pardinus permet aux utilisateurs de déclarer des bornes symboliques pour des relations mutables, afin que les dépendances sur les relations immuables puissent être rendues explicites.

4 Évaluation

Nous avons évalué la scalabilité de Pardinus pour les *backends* complets et bornés, la stratégie parallèle décomposée, et les opérations d'itération, avec de multiples variantes de 6 spécifications différentes. Le *backend* SAT semble mieux évoluer que SMV avec l'augmentation de la taille du modèle, en particulier pour les problèmes satisfaisables. Le *backend* SMV complet a été moins efficace mais se rapproche des performances des *backends* bornés à mesure que la longueur de trace maximale considérée augmente. Cela rend envisageable l'application d'une analyse complète lorsqu'une confiance suffisante est obtenue à partir des *backends* bornés. La stratégie parallèle montre des gains considérables pour les problèmes satisfaisables, en particulier pour les *backends* SMV. Les gains pour les problèmes UNSAT ne sont pas aussi cohérents, mais les backends SMV semblent en profiter davantage. Les deux opérations d'itération se sont révélées réalisables pour les sessions interactives avec les deux stratégies utilisées, bien que l'itération sur les configurations semble être affectée par le nombre de configurations valides. L'itération sur les configurations a été plus efficace dans l'approche non-parallélisée, tandis que l'itération de chemins s'est mieux comportée avec l'itération parallèle.

References

[1] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[2] J. Brunel, D. Chemouil, A. Cunha, and N. Macedo. The Electrum Analyzer: Model checking relational first-order temporal specifications. In *ASE*, pages 884–887. ACM, 2018.

[3] R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev. *NuSMV 2.6 User Manual*. FBK-IRST, 2010.

[4] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2nd edition, 2016.

[5] N. Macedo, J. Brunel, D. Chemouil, and A. Cunha. Pardinus: A temporal relational model finder. *Journal of Automated Reasoning*, 66:861–904, 2022.

[6] N. Macedo, J. Brunel, D. Chemouil, A. Cunha, and D. Kuperberg. Lightweight specification and analysis of dynamic systems with rich configurations. In *SIGSOFT FSE*, pages 373–383. ACM, 2016.

[7] K. Y. Rozier and M. Y. Vardi. LTL satisfiability checking. *STTT*, 12(2):123–137, 2010.

[8] A. Siegel, M. Santomauro, T. Dyer, T. Nelson, and S. Krishnamurthi. Prototyping formal methods tools: A protocol analysis case study. In *Protocols, Logic, and Strands: Essays Dedicated to Joshua Guttman on the Occasion of his 66.66th Birthday*, LNCS, 2021.

[9] E. Torlak and D. Jackson. Kodkod: A relational model finder. In *TACAS*, volume 4424 of *LNCS*, pages 632–647. Springer, 2007.

Un support efficace des critères de couverture de test avancés pour Klee *

Nicolas Berthier[1], Steven de Oliveira[1],
Nikolai Kosmatov[2], Delphine Longuet[2], Romain Soulat[2]

[1]OCamlPro, Paris, France
{nicolas.berthier, steven.de-oliveira}@ocamlpro.com

[2]Thales Research & Technology, Palaiseau, France
{nikolai.kosmatov, delphine.longuet, romain.soulat}@thalesgroup.com

Contexte et motivation

Les techniques de génération automatique de tests ont fait des progrès significatifs pendant les vingt dernières années. Un des succès les plus remarquables dans ce domaine est l'*exécution symbolique dynamique* (DSE) [2,3], une technique de génération de tests qui combine l'exécution symbolique et l'exécution concrète du programme sous test. Différents outils à base de la DSE ont été développés [4–11], et de nombreuses études de cas et applications industrielles ont été réalisées [7,8,10,12].

La DSE est basée sur une exploration des chemins d'exécution du programme sous test, avec pour but la couverture de tous les chemins. En général, la DSE est moins bien adaptée pour d'autres critères de couverture, tels que les décisions (branches), conditions, conditions multiples, mutations faibles, test aux limites, ou des objectifs de test fournis par l'utilisateur. En effet, différents critères nécessitent de couvrir des objectifs très différents, et leur support dans les outils reste limité.

Les travaux de Bardin *et al.* [13] ont proposé un mécanisme générique pour la spécification des critères de couverture de test à l'aide d'objectifs de test élémentaires, appelés *étiquettes de couverture* (ou *(coverage) labels*). Ces travaux ont également proposé des solutions pour une génération de tests efficace pour les labels. Cependant, ces techniques n'ont jamais été intégrées dans des générateurs de tests publiquement disponibles. La seule implémentation connue de ces techniques est leur intégration dans l'outil propriétaire PathCrawler [6], en boîte grise et en modifiant significativement la stratégie de l'outil [13]. Une évaluation industrielle de cette intégration a permis de montrer son efficacité [12]. L'absence d'un outil publiquement disponible avec un support large et efficace des critères de couverture de tests reste un obstacle important pour une utilisation plus large des critères. La principale motivation de ce travail consiste à lever cet obstacle.

Objectifs et contributions

Notre but est de démontrer qu'un support efficace des labels peut être intégré dans Klee [8], un générateur de tests populaire et ouvert, basé sur la DSE. Contrairement à l'intégration précédente dans PathCrawler [13], nous avons choisi une intégration en boîte noire qui pourra

*Cette soumission est un résumé étendu de l'article [1], publié à SAC-SVT 2023 (`https://arxiv.org/abs/2211.14592v2`).

servir d'inspiration pour une intégration légère des labels dans d'autres outils. La version de l'outil réalisée, appelée Klee4labels, est publiquement disponible [1].

Les contributions de ce travail sont les suivantes :

— une intégration, légère et non-intrusive, d'un support efficace pour différents critères de couverture de test avancés exprimés par des labels dans l'outil Klee ;
— une description détaillée de cette intégration sous forme d'une approche générique qui devrait être applicable également pour d'autres outils similaires ;
— une évaluation de la version étendue de Klee sur plusieurs exemples et une analyse détaillée des résultats qui confirme les avantages de la technique proposée.

Les résultats obtenus montrent que Klee4labels atteint une bien meilleure couverture des labels que Klee utilisé seul. Notre outil génère souvent moins de tests que Klee, et des tests plus pertinents car ils ciblent les labels. Enfin le temps de génération reste raisonnable comparé au temps d'exécution de Klee. Par exemple, sur le programme `gd_full_bad` (issu de la base d'exemples Verisec [14]), pour le critère de test aux limites, Klee couvre 32% des objectifs en 3,4 sec. en générant 33 tests, alors que Klee4labels couvre 84% des objectifs en 5,4 sec. en générant 16 tests.

Remerciement. Ces travaux ont été partiellement financés par le projet RAPID TAGAda de l'Agence de l'Innovation de Défence et par le projet EMASS de l'ANR. Les auteurs remercient les relecteurs pour leurs commentaires.

Références

[1] Nicolas Berthier, Steven de Oliveira, Nikolai Kosmatov, Delphine Longuet, and Romain Soulat. An efficient black-box support of advanced coverage criteria for Klee. In *SAC, Software Verification and Testing Track (SAC-SVT 2023)*. ACM, March 2023. To appear.

[2] Cristian Cadar, Patrice Godefroid, Sarfraz Khurshid, Corina S. Pasareanu, Koushik Sen, Nikolai Tillmann, and Willem Visser. Symbolic execution for software testing in practice : Preliminary assessment. In *ICSE*, pages 1066–1071, 2011.

[3] Cristian Cadar and Koushik Sen. Symbolic execution for software testing : Three decades later. *Commun. ACM*, 56(2) :82–90, 2013.

[4] Patrice Godefroid, Nils Klarlund, and Koushik Sen. DART : directed automated random testing. In *OSDI*, pages 213–223, 2005.

[5] Koushik Sen, Darko Marinov, and Gul Agha. CUTE : a concolic unit testing engine for C. In *FSE*, pages 263–272, 2005.

[6] Nicky Williams, Bruno Marre, Patricia Mouy, and Muriel Roger. PathCrawler : Automatic generation of path tests by combining static and dynamic analysis. In *EDCC*, volume 3463 of *LNCS*, pages 281–292, 2005.

[7] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE : automatically generating inputs of death. In *CCS*, pages 322–335, 2006.

[8] Cristian Cadar, Daniel Dunbar, and Dawson Engler. KLEE : Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, pages 209–224, 2008.

[9] Nikolai Tillmann and Jonathan de Halleux. Pex–white box test generation for .NET. In *TAP*, volume 4966 of *LNCS*, pages 134–153, 2008.

[10] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. Automated whitebox fuzz testing. In *NDSS*, 2008.

[11] Corina S Păsăreanu and Neha Rungta. Symbolic pathfinder : Symbolic execution of java bytecode. In *ASE*, pages 179–180, 2010.

1. `https://github.com/OCamlPro/klee4labels`

[12] Sébastien Bardin, Nikolai Kosmatov, Bruno Marre, David Mentré, and Nicky Williams. Test case generation with PathCrawler/LTest : How to automate an industrial testing process. In *ISOLA*, volume 11247 of *LNCS*, pages 104–120, 2018.

[13] Sébastien Bardin, Nikolai Kosmatov, and François Cheynier. Efficient leveraging of symbolic execution to advanced coverage criteria. In *ICST*, pages 173–182, 2014.

[14] Kelvin Ku, Thomas E. Hart, Marsha Chechik, and David Lie. A buffer overflow benchmark for software model checkers. In *ASE*, pages 389–392, 2007.

Nondeterministic, Recursive, and Impure Programs in Coq

Nicolas Chappe[1], Paul He [2], Ludovic Henrio [1], Yannick Zakowski[1], and Steve Zdancewic [2]

[1]Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP
[2]University of Pennsylvania

In [3], published this year at POPL'23, we are interested in developing tools for modeling nondeterministic computations in a dependent type theory such as Coq's CIC [13]. Many existent formalisms [2, 1, 10, 12] could be used to this end (and many have been [11, 4, 6, 5, 9]). Those techniques come with various tradeoffs when it comes to the needs of formalization. Many of these approaches are based on some kind of automata, or labeled transitions systems; while offering powerful bisimulation proof principles, they are not easily made modular. Relationally-defined operational semantics are flexible and expressive, but again suffer from issues of compositionality. Conversely, powerdomains and game semantics are more denotational approaches, aiming to ensure compositionality by construction; however, the mathematical structures involved are themselves very complex, typically involving many relations and constraints [1, 7, 10] that are not easy to implement in constructive logic (though there are some notable exceptions [5, 9]).

This paper introduces a new formalism[1] designed specifically to facilitate the definition of and reasoning about nondeterministic computations in Coq's dependent type theory. The key idea is to update Xia, et al.'s *interaction trees* (ITrees) framework [14] with native support for nondeterminisic "choice nodes" that represent internal choices made during computation.

The formal definition of the data-structure is given on Figure 1: computations are modelled as potentially infinite trees whose leaves (`Ret`) carry pure values of the return type `R`. Three different nodes are possible: *visible nodes* – `Vis` – model external interactions with the environment over an element of the signature of effects `E`, *stepping nodes* – `brS` – encode immediate internal nondeterministic branches, while *delay nodes* – `brD` – represent an internal non-deterministic branch that can only be taken by reaching further down an observable transition.

This intuition is embodied in the interpretation of the data structure into a labelled transition system (LTS), as defined in Figure 2: leaves, external interaction, and stepping branches directly map to a transition, while delayed branches look ahead in the structure inductively. We then transfer the well-established literature on simulations and bisimulations for LTSs to CTrees. In particular, we define the core notion of equivalence of computations represented as CTrees as strong bisimulation.

We prove that equipped with this (strong bisimulation) equivalence, CTrees have the structure of an iterative monad. Furthermore, a monad morphism from ITrees into CTrees makes them an adequate monad into which non-deterministic events can be interpreted. Additional effects, such as stateful ones, can still be cleanly handled, as we show that interpretation from CTrees into arbitrary iterative monads is suitable. Additionally, internal branching nodes can be themselves implemented, leading to a refinement —a simulation— of the computation.

We demonstrate the use of our structure on two main case studies. We give a model to `ccs` [8], establishing its traditional (strongly bisimilar) equational theory and up-to contexts, and proving that the equivalence we obtain coincides with the usual operational strong bisimilarity. We show how to mix stateful effects with non-determinism by providing a model for a language with cooperative multithreading.

[1]All our results are formalized in Coq and available online: https://github.com/vellvm/ctrees

```
CoInductive ctree (E : Type → Type) (R : Type) :=
| Ret (r : R)
| Vis {X : Type} (e : E X) (k : X → ctree)
| brS (n : nat) (k : fin n → ctree)
| brD (n : nat) (k : fin n → ctree)
```

Figure 1: CTrees: definition

$$\texttt{label} \triangleq \texttt{tau} \mid \texttt{obs}\ e\ v \mid \texttt{val}\ v$$

$$\frac{k\ v \xrightarrow{l} t}{Br^n_D\ k \xrightarrow{l} t} \qquad \frac{}{Br^n_S\ k \xrightarrow{\texttt{tau}} k\ v} \qquad \frac{}{Vis\ e\ k \xrightarrow{\texttt{obs}\ e\ v} k\ v} \qquad \frac{}{\texttt{Ret}\ v \xrightarrow{\texttt{val}\ v} \emptyset}$$

Figure 2: Inductive characterisation of the LTS induced by a CTree

References

[1] Sampson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *Proceedings of the 14th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1999.

[2] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science, 2001.

[3] Nicolas Chappe, Paul He, Ludovic Henrio, Yannick Zakowski, and Steve Zdancewic. Choice trees: Representing nondeterministic, recursive, and impure programs in coq. *Proc. ACM Program. Lang.*, 7(POPL):1770–1800, 2023.

[4] Jeehoon Kang, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis, and Derek Dreyer. A promising semantics for relaxed-memory concurrency. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 175–189. ACM, 2017.

[5] Jérémie Koenig and Zhong Shao. Refinement-based game semantics for certified abstraction layers. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 633–647, New York, NY, USA, 2020. Association for Computing Machinery.

[6] Sung-Hwan Lee, Minki Cho, Anton Podkopaev, Soham Chakraborty, Chung-Kil Hur, Ori Lahav, and Viktor Vafeiadis. Promising 2.0: global optimizations in relaxed memory concurrency. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 362–376. ACM, 2020.

[7] Paul-André Melliès and Samuel Mimram. Asynchronous games: Innocence without alternation. In Luís Caires and Vasco T. Vasconcelos, editors, *CONCUR 2007 – Concurrency Theory*, pages 395–411, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[8] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., USA, 1989.

[9] Arthur Oliveira Vale, Paul-André Melliès, Zhong Shao, Jérémie Koenig, and Léo Stefanesco. Layered and object-based game semantics. *Proc. ACM Program. Lang.*, 6(POPL), jan 2022.

[10] Silvain Rideau and Glynn Winskel. Concurrent strategies. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 409–418. IEEE Computer Society, 2011.

[11] Jaroslav Sevcík, Viktor Vafeiadis, Francesco Zappa Nardelli, Suresh Jagannathan, and Peter Sewell. Compcerttso: A verified compiler for relaxed-memory concurrency. *J. ACM*, 60(3):22:1–22:50, 2013.

[12] M.B. Smyth. Powerdomains. In *Mathematical Foundations of Computer Science*, volume 4 of *Lecture Notes in Computer Science*. Springer, 1976.

[13] The Coq Development Team. The coq proof assistant, January 2022.

[14] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. Interaction trees. *Proceedings of the ACM on Programming Languages*, 4(POPL), 2020.

Génération automatique de tests d'égalité corrects en Coq, en pratique

Benjamin Grégoire Jean-Christophe Léchenet
Enrico Tassi
Université Côte d'Azur, Inria, Sophia Antipolis, France

Le compilateur Jasmin, principalement écrit en Coq, s'appuie sur la bibliothèque Mathematical Components, et notamment son interface `eqType`, qui désigne un type équipé d'un test d'égalité correct, c'est-à-dire une fonction booléenne démontrée équivalente à l'égalité logique sur ce type. Pour utiliser cette interface sur un type inductif concret de Jasmin, il est nécessaire de définir un test d'égalité et de le démontrer correct pour ce type en particulier.

C'est une tâche plus fastidieuse que complexe, et on aimerait utiliser un outil de génération automatique. Hélas, les outils actuels, qu'ils viennent avec Coq ou qu'ils soient mis à disposition par des bibliothèques externes, souffrent de deux types de problèmes. D'une part, ils ont un problème de performance : ils mettent un temps déraisonnable sur des types un peu gros, par exemple sur les types inductifs à plus de 100 constructeurs (dans Jasmin, le type des instructions assembleurs x86 est de cet ordre de grandeur). D'autre part, ils ont un problème d'expressivité. Typiquement, ils ne gèrent pas bien l'utilisation de conteneurs ou les types dépendants. Cela conduit à devoir écrire manuellement certains tests d'égalité et leurs preuves.

Le problème de performance n'est pas inattendu : la manière naturelle d'écrire un test d'égalité sur un type inductif est de faire un double pattern-matching sur les arguments, ce qui conduit à un code de taille quadratique en le nombre de constructeurs du type inductif. Il faut donc utiliser une autre formulation des tests d'égalité, plus consise.

Dans ce travail, nous proposons un schéma pour définir des tests d'égalité et les preuves associées de taille proportionnelle à la taille du type inductif considéré. Ce schéma supporte la notion de conteneur et sait gérer une certaine forme de types dépendants. Nous avons implémenté ce schéma sous la forme d'un outil appelé `feqb`, écrit en Coq-Elpi, un langage de script pour Coq. Cet outil a été intégré dans la version 1.16.0 de Coq-Elpi et remplace l'ancien outil de génération automatique qui était fourni par Coq-Elpi. Des tests sur des types inductifs à plusieurs centaines de constructeurs confirment que `feqb` est bien plus rapide que les outils préexistants.

Cet article est un résumé étendu de l'article "Practical and Sound Equality Tests, Automatically : Deriving eqType Instances for Jasmin's Data Types with

Coq-Elpi", accepté à CPP 2023 (12th ACM SIGPLAN International Conference on Certified Programs and Proofs).

Energy Büchi Problems *

Sven Dziadek[0000−0001−6767−7751], Uli Fahrenberg[0000−0001−9094−7625], and Philipp Schlehuber-Caissier[0000−0002−6611−9659]

EPITA Research Laboratory (LRE), France

Abstract. We show how to efficiently solve energy Büchi problems in finite weighted automata and in one-clock weighted timed automata. Solving the former problem is our main contribution and is handled by a modified version of Bellman-Ford interleaved with Couvreur's algorithm. The latter problem is handled via a reduction to the former relying on the corner-point abstraction. All our algorithms are freely available and implemented in a tool based on the open source frameworks TChecker and Spot.

Keywords: weighted timed automaton; weighted automaton; energy problem; generalized Büchi acceptance; energy constraints

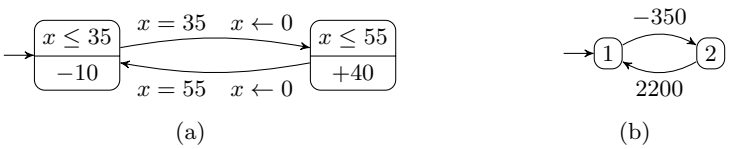
Energy problems in weighted (timed) automata pose the question whether there exist infinite runs in which the accumulated weights always stay positive. Since their introduction in [5], much research has gone into different variants of these problems, for example energy games [8, 11, 17], energy parity games [7], robust energy problems [2, 3], etc., and into their application in embedded systems [12,13], satellite control [4,16], and other areas. Nevertheless, many basic questions remain open and implementations are somewhat lacking.

The above results discuss *looping* automata [18], *i.e.,* ω-automata which accept any infinite word. In practice, looping automata do not suffice because they cannot express all liveness properties. For model checking, formal properties (*e.g.,* in LTL) are commonly translated into (generalized) Büchi automata [6] that provide a simple model for the larger class of ω-regular languages.

In this work, we extend energy problems with transition-based generalized Büchi conditions and treat them for weighted automata as well as weighted timed automata with precisely one clock. On weighted automata we show that they are effectively decidable using a combination of a modified Bellman-Ford algorithm with Couvreur's algorithm. For weighted timed automata we show that one can use the corner-point abstraction to translate the problem to weighted (untimed) automata.

For looping automata, the above problems have been solved in [5]. (This paper also treats energy games and so-called universal energy problems, both of which are of no concern to us here.) While we can re-use some of the methods of

* Partially funded by ANR project Ticktak (ANR-18-CE40-0015). Based on a paper [10] presented at the 25th International Symposium on Formal Methods, FM'23.



(a) (b)

Fig. 1: Satellite example: two representations of the base circuit. (a) as weighted timed automaton A; (b) as a (finite) weighted automaton.

[5] for our Büchi-enriched case, our extension is by no means trivial. First, in the setting of [5] it suffices to find any reachable and energy positive loop; now, our algorithm must consider that such loops might not be accepting in themselves but give access to new parts of the automaton which are. Second, [5] mostly treat the energy problem with unlimited upper bound, whereas we consider that energy has a ("weak") upper bound beyond which it cannot increase. [5] claim that the weak-upper-bound problem can be solved by slight modifications to their solution of the unbounded problem; but this is not the case. For example, the typical Bellman-Ford detection of positive cycles might not work when the energy levels attained in the previous step are already equal to the upper bound.

As a second contribution, we have implemented all of our algorithms in a tool based on the open-source platforms TChecker[1] [15] and Spot[2] [9] to solve generalized energy Büchi problems for one-clock weighted timed automata. We first employ TChecker to compute the zone graph and then use this to construct the corner-point abstraction. This in turn is a weighted (untimed) generalized Büchi automaton, in which we also may apply a variant of Alur and Dill's Zeno-exclusion technique [1]. Finally, our main algorithm to solve generalized energy Büchi problems on weighted finite automata is implemented using a fork of Spot. Our software is available at `https://github.com/PhilippSchlehuberCaissier/wspot`.

In our approach to solve the latter problem, we do not fully separate the energy and Büchi conditions (contrary to, for example, [7] who reduce energy parity games to energy games). We first determine the strongly connected components (SCCs) of the unweighted automaton. Then we degeneralize each Büchi accepting SCC one by one, using the standard counting construction [14]. Finally, we apply a modified Bellman-Ford algorithm to search for energy feasible lassos that start on the main graph and loop in the SCC traversing the remaining Büchi condition.

Example 1. A satellite in low-earth orbit has a rotation time of about 90 minutes, 40% of which are spent in earth shadow. Measuring time in minutes and (electrical) energy in unspecified "energy units", we may thus model its simplified base electrical system as shown in Fig. 1a.

This is a weighted timed automaton with one clock, x, and two locations. The clock is used to model time, which progresses with a constant rate but can be reset on transitions. The initial location on the left (modeling earth shadow) is

[1] See `https://github.com/ticktac-project/tchecker`
[2] See `https://spot.lrde.epita.fr/`

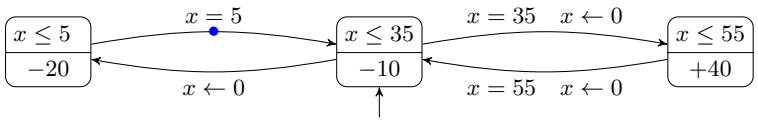


Fig. 2: Weighted timed automaton A_1 for satellite with work module.

only active as long as $x \leq 35$, and given that x is initially zero, this means that the model may stay here for at most 35 minutes. Staying in this location consumes 10 energy units per minute, corresponding to the satellite's base consumption.

After 35 minutes the model transitions to the "sun" location on the right, where it can stay for at most 55 minutes and the solar panels produce 50 energy units per minute, from which the base consumption has to be subtracted. Note that the transitions can only be taken if the clock shows exactly 35 (resp. 55) minutes and is reset to zero after the transition as denoted by $x \leftarrow 0$. This ensures that the satellite stays exactly 35 minutes in the shadow and 55 minutes in the sun, roughly consistent with the "physical" model.

Figure 1b shows a translation of the automaton of Fig. 1a to a weighted untimed automaton. State 1 corresponds to the "shadow" location, transitions are annotated with the corresponding weights, the rate of the location multiplied by the time spend in it. We give an algorithm to obtain a weighted automaton from a weighted timed automaton with precisely one clock.

One may now pose the following question: for a given battery capacity b and an initial charge c, is it possible for the satellite to function indefinitely without ever running out of energy? It is clear that for $c < 350$ or $b < 350$, the answer is no: the satellite will run out of battery before ever leaving Earth's shadow; for $b \geq 350$ and $c \geq 350$, it will indeed never run out of energy.

Now assume that the satellite also has some work to do: once in a while it must, for example, send some collected data to earth. Given that we can only handle weighted automata with precisely one clock, we model the combined system as in Fig. 2. That is, work (modeled by the leftmost location) takes 5 minutes and costs an extra 10 energy units per minute. The colored dot on the outgoing transition of the work state marks a (transition-based) Büchi condition which forces us to see the transition infinitely often in order for the run to be accepted. As a consequence, all accepting runs also visit the "work" state indefinitely often, consistent with the demand to send data once in a while. In order to model the system within the constraints of our modeling formalism, we must make two simplifying assumptions, both unrealistic but conservative:

- work occurs during earth shadow;
- work prolongs earth shadow time.

The reason for the second property is that the clock x is reset to 0 when entering the work state; otherwise we would not be able to model that it lasts 5 minutes without introducing a second clock. It is clear how further work modules may be added in a similar way, each with their own accepting color.

References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
2. Giovanni Bacci, Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Pierre-Alain Reynier. Optimal and robust controller synthesis - using energy timed automata with uncertainty. In Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik P. de Vink, editors, *FM*, volume 10951 of *Lect. Notes Comput. Sci.*, pages 203–221. Springer, 2018. Best Paper award.
3. Giovanni Bacci, Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Pierre-Alain Reynier. Optimal and robust controller synthesis using energy timed automata with uncertainty. *Formal Aspects Comput.*, 33(1):3–25, 2021.
4. Morten Bisgaard, David Gerhardt, Holger Hermanns, Jan Krčál, Gilles Nies, and Marvin Stenger. Battery-aware scheduling in low orbit: The GomX-3 case. In John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou, editors, *FM*, volume 9995 of *Lect. Notes Comput. Sci.*, pages 559–576. Springer, 2016.
5. Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *FORMATS*, volume 5215 of *Lect. Notes Comput. Sci.*, pages 33–47. Springer, 2008.
6. J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1966.
7. Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012.
8. Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 8 of *LIPIcs*, pages 505–516, 2010.
9. Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and *omega*-automata manipulation. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *ATVA*, volume 9938 of *Lect. Notes Comput. Sci.*, pages 122–129. Springer, 2016.
10. Sven Dziadek, Uli Fahrenberg, and Philipp Schlehuber-Caissier. Energy büchi problems. In Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker, editors, *FM*, volume 14000 of *Lect. Notes Comput. Sci.*, pages 222–239. Springer, 2023.
11. Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jiří Srba. Energy games in multiweighted automata. In Antonio Cerone and Pekka Pihlajasaari, editors, *ICTAC*, volume 6916 of *Lect. Notes Comput. Sci.*, pages 95–115. Springer, 2011.
12. Heiko Falk, Kevin Hammond, Kim G. Larsen, Björn Lisper, and Stefan M. Petters. Code-level timing analysis of embedded software. In Ahmed Jerraya, Luca P. Carloni, Florence Maraninchi, and John Regehr, editors, *EMSOFT*, pages 163–164. ACM, 2012.
13. Goran Frehse, Kim G. Larsen, Marius Mikučionis, and Brian Nielsen. Monitoring dynamical signals while testing timed aspects of a system. In Burkhart Wolff and Fatiha Zaïdi, editors, *ICTSS*, volume 7019 of *Lect. Notes Comput. Sci.*, pages 115–130. Springer, 2011.
14. Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lect. Notes Comput. Sci.*, pages 53–65. Springer, 2001.

15. Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Inf. Comput.*, 251:67–90, 2016.
16. Marius Mikučionis, Kim G. Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. Schedulability analysis using Uppaal: Herschel-Planck case study. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA (2)*, volume 6416 of *Lect. Notes Comput. Sci.*, pages 175–190. Springer, 2010.
17. Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015.
18. Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths (extended abstract). In *FOCS*, pages 185–194. IEEE Computer Society, 1983.

Vérification de propriétés interactives sur des systèmes interactifs

Cécile Marcon[1,2], Xavier Thirioux[1], Celia Picard[2], Cyril Allignol[2]
ISAE SUPAERO[1] / ENAC[2], Toulouse, France

Résumé

Les systèmes interactifs sont des systèmes informatiques qui échangent avec un utilisateur humain. Ils peuvent être décrits grâce à un UIDL (User Interface Description Language). Le sujet de thèse décrit dans cet article vise à apporter des garanties sur de tels systèmes. À cette fin, nous voulons exprimer et vérifier formellement des propriétés qui portent sur des aspects interactifs de programmes décrits à l'aide d'un UIDL. Nous disposons pour cela d'un UIDL formel, BIGUIL, dont la sémantique est décrite à l'aide des bigraphes. Nous voulons explorer l'outillage de BIGUIL pour garantir ces propriétés par construction.

Keywords : interactive properties, interactive systems, formal expression

1 Introduction : vérifier les systèmes interactifs

L'objectif de la vérification des systèmes est de fournir des garanties aux utilisateurs d'un système. En effet, il est nécessaire de fournir des garanties lorsque l'on travaille avec des systèmes critiques. Les systèmes critiques interactifs ne sont pas différents des systèmes critiques non interactifs en termes de rigueur et d'exigences. Cependant, les interactions avec les humains introduisent un nouvel ensemble de propriétés appelées propriétés interactives.

Une propriété interactive est une propriété qui définit les exigences du système en réponse aux interactions de l'utilisateur [1]. Par exemple, on peut exiger qu'une information ne soit jamais recouverte par une fenêtre contextuelle ou qu'un click sur un bouton ferme une fenêtre. Dans la littérature, des travaux concernent la vérification de propriétés interactives. Ainsi, Oliveira, Palanque et Weyers [2] et Béger [3] recensent ces travaux de façon à lister les propriétés interactives ainsi que les différentes approches adoptées pour les vérifier.

La littérature propose également des formalismes comme la logique de Hoare [4] ou encore la logique temporelle [5] pour formaliser une partie des propriétés interactives. Des approches comme l'analyse statique [6], le model checking [5] ou la preuve de théorème [4] permettent de prouver certaines propriétés interactives sur les interfaces, le noyau fonctionnel etc. Cependant, nous souhaitons explorer la

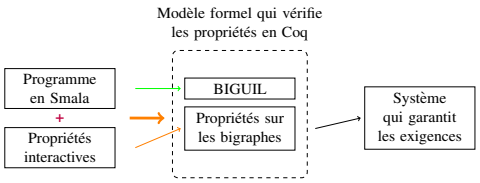


Figure 1: Approche de la thèse

création d'un framework assurant l'expression des propriétés interactives par des non professionnels des preuves formelles jusqu'à la génération d'un système les garantissant. Notre approche se base sur un modèle formel BIGUIL [7] (présenté brièvement Section 3) que nous souhaitons explorer davantage car il reflète plus fidèlement la structure des interfaces utilisateurs que d'autres modèles existants.

Le schéma de la Figure 1 décrit notre approche. Nous souhaitons permettre à un développeur d'exprimer les propriétés interactives dans un UIDL (langage de description d'interface). Notre UIDL d'étude est le langage Smala [8] développé à l'ENAC. Un premier objectif est donc d'exprimer les propriétés interactives d'une application codée en Smala en relation avec son code source. Nous souhaitons compiler le code muni de ces propriétés interactives vers un modèle formel qui vérifie les propriétés. Nous possédons un UIDL formel proposé par Nalpon [7], BIGUIL, dont la sémantique repose sur les bigraphes. Smala peut être compilé vers une implémentation de BIGUIL en OCaml qui utilise notamment l'API BigraphER [9]. Nos travaux se concentreront principalement sur la traduction des propriétés interactives en propriétés sur les bigraphes et sur la création du modèle formel qui garantit les propriétés avec tous ces outils.

2 Identifier les propriétés interactives

Nous cherchons à représenter formellement les propriétés interactives afin de pouvoir les prouver. La première étape est d'identifier et lister les propriétés interactives.

L'état de l'art [2], [3] nous a permis d'identifier les principaux thèmes de propriétés interactives (nous les détaillons dans la suite avec les résultats de notre approche). Nous avons mené une approche centrée utilisateurs telle que décrite par Beaudouin et al. [10] afin d'identifier les besoins réels des développeurs. La démarche centrée utilisateur est une approche qui implique l'utilisateur final du système dans chaque phase de conception du système au travers d'ateliers. L'application de cette approche a pour but de rendre le produit final plus utilisable. Dans notre approche centrée utilisateur, nos utilisateurs sont les développeurs de systèmes interactifs et le système que nous concevons est un système qui leur permettrait d'exprimer des propriétés interactives qui seront garanties dans leurs applications. Nous avons notamment mené six entretiens contextuels (discussions guidées autour de l'utilisation d'un produit par son utilisateur dans son environnement réel) avec des développeurs professionnels de systèmes interactifs.

Nos résultats montrent que parmi les propriétés interactives, nous retrouvons chacune des trois catégories proposées par Campos et Harrison dans [11]:

- visibilité : des propriétés statiques qui caractérisent une interface. Il peut s'agir de description des aspects graphiques et autres modalités de l'interface, comme de propriétés plus complexes comme la perciptibilité de composants et toutes les questions difficiles que cela apporte (i.e. facteurs humains) ;
- atteignabilité : des propriétés d'interaction qui caractérisent les fonctionnalités de l'interfaces. Par exemple les réactions d'un composant à des évènements, mais cette catégorie inclue également l'existence de fonctionnalités ou des propriétés temporelles telles que le temps de réaction ;
- fiabilité : des propriétés d'intégration exprimées au niveau du système, comme par exemple la véracité d'un prédicat au travers des modes ou états.

Les propriétés les plus élaborées peuvent être décomposées en propriétés plus élémentaires par leurs concepteurs. Ainsi, un concepteur souhaitant exprimer qu'un composant est *visible* pourrait exprimer que le composant doit être d'une certaine taille, qu'il n'est pas masqué par un autre composant... C'est pourquoi nous nous intéressons aux propriétés élémentaires sur lesquelles nous pourrons construire des exemples plus complexes.

Nous voulons maintenant explorer la possibilité de garantir des propriétés interactives sur un système interactif. Pour cela, nous voulons compiler un programme muni de ces propriétés interactives en un modèle formel qui les assure.

3 Travaux en cours : formaliser les propriétés interactives

L'objectif est de formaliser et prouver ces propriétés. Nous devons répondre à deux questions concernant la formalisation : comment exprimer les propriétés dans le code, pour des développeurs non experts en vérification formelle, et comment formaliser les propriétés en vue de la preuve.

Pour l'expression des propriétés par les développeurs, nous suivons une approche centrée sur les utilisateurs. Nous étudions notamment l'utilisation de langages d'annotation directement dans le code.

Quant à la formalisation des propriétés, nous les traduirons en propriétés sur les bigraphes. En effet, notre framework inclue BIGUIL introduit par Nalpon et al. [7], un UIDL formel basé sur les bigraphes. Un bigraphe, introduit par Milner [12], est composé d'une paire de graphes : le graphe de places, qui représente les relations spatiales entre les composants sous forme de forêt, et le graphe de liens, un hypergraphe représentant les liaisons entre les composants. À cela s'ajoute la notion de règle de réécriture, qui permet de remplacer une partie d'un bigraphe par un autre, introduisant ainsi de la dynamique.

Notre volonté d'inclure BIGUIL dans notre framework vient du fait que ces structures mathématiques sont particulièrement bien adaptées pour modéliser aussi

bien les aspects structurels (imbrication, positionnement) que les aspects dynamiques (activation de composants, actions utilisateur) des systèmes interactifs. Nous souhaitons représenter la sémantique de programme grâce à BIGUIL dans Coq pour y prouver des propriétés. Nos travaux actuels concernent l'outillage de la théorie des bigraphes en Coq en y décrivant les structures de base et opérations élémentaires des bigraphes.

References

[1] G. D. Abowd, J. Coutaz, and L. Nigay, "Structuring the space of interactive system properties.," *Engineering for Human-Computer Interaction*, vol. 18, pp. 113–129, 1992.

[2] R. Oliveira, P. Palanque, B. Weyers, *et al.*, "State of the art on formal methods for interactive systems," *The Handbook of formal methods in human-computer interaction*, pp. 3–55, 2017.

[3] D. Prun and P. Béger, "Formal verification of graphical properties of interactive systems," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. EICS, pp. 1–30, 2022.

[4] P. Bumbulis, P. S. C. Alencar, D. D. Cowan, *et al.*, "Validating properties of component-based graphical user interfaces," in *Design, Specification and Verification of Interactive Systems '96*, Vienna: Springer Vienna, 1996.

[5] B. d'Ausbourg, "Using model checking for the automatic validation of user interfaces systems," in *Design, Specification and Verification of Interactive Systems '98*, P. Markopoulos and P. Johnson, Eds., 1998, pp. 242–260.

[6] S. Chatty, M. Magnaudet, and D. Prun, "Verification of properties of interactive components from their executable code," in *Proceedings of the 7th ACM SIGCHI Symposium on EICS 15*, 2015.

[7] N. Nalpon, C. Allignol, and C. Picard, "Towards a user interface description language based on bigraphs," in *Theoretical Aspects of Computing*, 2022.

[8] M. Magnaudet, S. Chatty, S. Conversy, *et al.*, "Djnn/smala: A conceptual framework and a language for interaction-oriented programming," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. EICS, 2018.

[9] M. Sevegnani and M. Calder, "Bigrapher: Rewriting and analysis engine for bigraphs," in *Computer Aided Verification*, 2016, pp. 494–501.

[10] M. Beaudouin-Lafon and W. E. Mackay, "Prototyping tools and techniques," in *Human-Computer Interaction*, CRC Press, 2009, pp. 137–160.

[11] J. C. Campos and M. D. Harrison, "Formally verifying interactive systems," in *Design, Specification and Verification of Interactive Systems '97*, 1997.

[12] R. Milner, *The space and motion of communicating agents*. Cambridge University Press, 2009.

Cybersécurité pour les systèmes embarqués critiques à base d'Intelligence Artificielle

Céline Bellanger
ENAC, Toulouse

Abstract

L'intelligence artificielle est de plus en plus utilisée dans les systèmes embarqués critiques, et particulièrement dans l'aéronautique. Elle peut remplacer des fonctions existantes comme la stabilisation ou le guidage ; ou ouvrir de nouvelles possibilités comme la réalisation de toutes les phases de vol de façon autonome notamment grâce à la reconnaissance d'images. Cependant, elle peut être la cible de nouveaux types de cyberattaques, spécifiques à l'intelligence artificielle. Dans ce document, nous présentons nos travaux en cours et à venir pour améliorer la sûreté des réseaux de neurones utilisés dans les systèmes embarqués. Nous nous appuierons sur des méthodes formelles pour étudier les propriétés temporelles des signaux issus des réseaux de neurones. L'approche visera à 1. définir les méthodes et outils pour évaluer les propriétés exprimées en Signal Temporal Logic (STL), 2. caractériser les propriétés d'intérêt dans cette logique STL et 3. étudier la validité de contrôleurs à base de réseaux de neurones vis-à-vis de ces propriétés.

1 Introduction

Le développement de l'intelligence artificielle joue un rôle de plus en plus important pour l'industrie aéronautique et aérospatiale ces dernières années. Elle est aussi bien employée dans les systèmes embarqués pour effectuer des fonctions de contrôle ou de stabilisation [4], que pour des tâches nécessitant jusqu'alors l'intervention humaine comme le décollage ou l'atterrissage des avions [1].

Bien que les premiers résultats soient encourageants, ces avancées récentes questionnent notre capacité à certifier l'intelligence artificielle déployée dans les systèmes embarqués critiques. En particulier, des cyberattaques spécifiques aux réseaux de neurones sont susceptibles d'atteindre ces systèmes pour provoquer un comportement indésirable. C'est notamment le cas des *adversarial attacks* qui altèrent légèrement les images fournies en entrée du système pour impacter fortement la classification obtenue [9].

Notre objectif est d'améliorer la sûreté des contrôleurs basés sur des réseaux de neurones, dans les systèmes critiques. Les méthodes formelles assurent qu'un système répond en toutes circonstances à certaines propriétés. Nous voulons les employer ici comme outil de vérification des réseaux neuronaux, dans le but de démontrer leur résilience en cas d'attaque. Nous utiliserons particulièrement la logique temporelle de signaux (STL ou *Signal Temporal Logic*) permettant l'analyse de l'évolution de la valeur des signaux en fonction du temps. Pour ce faire, nous aurons besoin d'identifier les propriétés d'intérêt pour l'étude du comportement des réseaux neuronaux, et de créer les solutions permettant d'évaluer ces propriétés au sein des contrôleurs. Des outils d'analyse statique tels que le model checking ou l'interprétation abstraite pourront ensuite valider les propriétés.

Dans cet article, nous présentons notre approche pour améliorer la sûreté des réseaux de neurones, et notre confiance dans les résultats qu'ils fournissent. Les sections suivantes introduisent les objectifs que nous souhaitons atteindre au cours de ma thèse, pour lesquels nous n'avons à ce jour que des résultats très préliminaires. La section 2 détaille les types de propriétés que nous voulons étudier. La section 3 décrit les outils de vérification STL en cours de développement afin d'analyser les propriétés identifiées.

2 Définition de propriétés sur les réseaux neuronaux

Plusieurs types de propriétés nous intéresseront pour étudier les contrôleurs à base de réseaux neuronaux présents dans les aéronefs.

Premièrement, nous devrons nous assurer que les propriétés fonctionnelles concernant les contrôleurs sont respectées. Il s'agit par exemple de garantir la stabilité, la performance, la robustesse de la stabilité, et la robustesse de la performance d'un aéronef.

Par la suite, nous nous intéresserons plus spécifiquement aux propriétés fonctionnelles des contrôleurs basés sur des réseaux de neurones. Il s'agira de définir des propriétés à satisfaire, ou au contraire à ne pas satisfaire, liées :

- à l'environnement : s'assurer que les données entrant dans le réseau de neurones sont conformes, pour éviter par exemple les *adversarial attacks* ;
- aux données d'entraînement et de validation du système : s'assurer que les données ne présentent pas de valeurs aberrantes, qui auraient pu être corrompues dans un but malveillant pour dégrader l'entraînement du réseau neuronal ;
- à la structure du réseau de neurones en lui-même : selon le type de réseau de neurones, il est possible d'extrapoler des propriétés de certaines couches ou certains noeuds du réseau.

Enfin, nous nous intéresserons aux propriétés propres aux cyberattaques sur les réseaux de neurones et les systèmes embarqués. La détermination de celles-ci passe par l'étude des cyberattaques spécifiques aux réseaux de neurones répertoriées à ce jour, notamment dans la matrice fournie par l'organisme MITRE [7].

Nous voulons exprimer toutes les propriétés fonctionnelles identifiées sous la forme de spécifications formelles. Une piste intéressante pour cela est la logique temporelle STL. Elle est utilisée pour décrire le comportement de signaux continus ou discrets sur un intervalle de temps borné. Nous pensons pouvoir représenter un grand nombre de ces propriétés sous la forme de propriétés STL. Ce travail a été initié par Kapinsky [5] dans le cas des propriétés liées aux contrôleurs.

Actuellement, les outils d'étude de propriétés STL ne permettent pas d'exprimer ces propriétés sont la forme d'observateurs synchrones et ne permettent donc pas de les étudier avec des outils de model-checking de langages synchrones, comme Kind2 pour Lustre.

Par conséquent, il est nécessaire de définir nos propres outils. C'est l'objet de la section suivante.

3 Vers une vérification exhaustive de propriétés STL

Des travaux se penchent déjà sur la génération de moniteurs basés sur des propriétés STL [2, 8].

Cependant, l'absence de preuve formelle de la correction des opérateurs STL proposés par rapport à leur définition théorique est un frein à leur utilisation dans le cadre de systèmes embarqués critiques.

Par ailleurs, les résultats présentés indiquent si les propriétés à analyser sont satisfaites ou non, mais ne permettent pas de représenter l'indétermination, c'est-à-dire les situations pour lesquelles, à un instant t, il n'est pas encore possible de conclure quant à la vérification ou la violation d'une propriété donnée.

Afin de représenter cette indétermination, nous avons pour objectif de définir une sémantique de STL à base d'observateurs synchrones. Elle reposera sur la logique trivaluée décrite par Kleene [6]. Ainsi à tout instant, une propriété STL est soit satisfaite, soit non satisfaite, soit (encore) indéterminée.

Soit $\mathcal{X}$ un signal, et φ, φ_1, φ_2 des formules STL [3]. Nous nous intéresserons aux opérateurs suivants :

- $\Diamond_{[a,b]}\varphi$ (Eventually) est satisfait lorsque la propriété φ est satisfaite au moins une fois dans $[a, b]$: $\exists t' \in t + [a, b] : (\mathcal{X}, t') \vDash \varphi$
- $\Box_{[a,b]}\varphi$ (Always) est satisfait lorsqu'à tout instant dans $[a, b]$ la propriété φ est satisfaite : $\forall t' \in t + [a, b] : (\mathcal{X}, t') \vDash \varphi$

- $\varphi_1 \, \mathcal{U}_{[a,b]} \, \varphi_2$ (Until) est satisfait lorsque φ_2 est satisfait à un instant dans l'intervalle $[a, b]$ et ssi, jusqu'à cet instant compris, φ_1 était toujours satisfait : $\exists t' \in t + [a, b] : (\mathcal{X}, t') \vDash \varphi_2 \wedge \forall t'' \in [t, t'] : (\mathcal{X}, t'') \vDash \varphi_1$

Un premier fragment consistera à proposer une version basée sur des observateurs synchrones pour chacun de ces opérateurs. Nous démontrerons formellement leur correction vis-à-vis de la définition théorique des opérateurs décrite ci-dessus, en établissant par model checking l'absence de contre-exemple. Dans un second temps, nous ajouterons l'imbrication d'opérateurs pour étudier des propriétés STL plus complexes. Par exemple, il serait possible de représenter la propriété suivante $\Diamond_{[a,b]}(\Box_{[0,2]}\varphi)$ qui exprime qu'il existe au moins un instant dans $[a, b]$ pour lequel la propriété φ est satisfaite à cet instant précis puis aux deux instants suivants.

Ces outils nous permettront de nous assurer que les réseaux de neurones à étudier satisfont effectivement les propriétés STL qu'ils sont censés vérifier.

4 Conclusion

Cet article résume notre approche pour améliorer la cybersécurité des réseaux de neurones utilisés dans les aéronefs critiques tels que les avions, les fusées ou les drones. Il décrit le type de propriétés que nous voulons étudier sur les réseaux de neurones, et les outils que nous devrons mettre en place pour y parvenir. À ce jour, nous avons initié le développement d'outils pour évaluer des propriétés STL basées sur des opérateurs non imbriqués. Nous poursuivrons ces travaux en travaillant à la représentation de formules STL plus complexes, s'appuyant sur des opérateurs STL imbriqués.

References

[1] *Airbus concludes ATTOL with fully autonomous flight tests — Airbus*. Section: Innovation. Oct. 28, 2021. URL: https://www.airbus.com/en/newsroom/press-releases/2020-06-airbus-concludes-attol-with-fully-autonomous-flight-tests (visited on 02/20/2023).

[2] Alessio Balsini et al. "Generation of simulink monitors for control applications from formal requirements". In: *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*. Toulouse, June 2017, pp. 1–9. DOI: 10.1109/SIES.2017.7993389.

[3] Alexandre Donzé. "On Signal Temporal Logic". In: *Runtime Verification*. Ed. by Axel Legay and Saddek Bensalem. Ed. by David Hutchison et al. Vol. 8174. Springer, 2013, pp. 382–383. DOI: 10.1007/978-3-642-40787-1_27.

[4] Jemin Hwangbo et al. "Control of a Quadrotor with Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 2.4 (Oct. 2017), pp. 2096–2103. DOI: 10.1109/LRA.2017.2720851. arXiv: 1707.05110[cs].

[5] James Kapinski et al. "ST-Lib: A Library for Specifying and Classifying Model Behaviors". In: Apr. 5, 2016. DOI: 10.4271/2016-01-0621.

[6] Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland. Amsterdam, 1952.

[7] *MITRE — ATLAS™*. URL: https://atlas.mitre.org/ (visited on 04/02/2023).

[8] Akshay Rajhans et al. "Specification and Runtime Verification of Temporal Assessments in Simulink". In: *Runtime Verification*. Ed. by Lu Feng and Dana Fisman. Vol. 12974. LNCS. Springer, 2021, pp. 288–296. DOI: 10.1007/978-3-030-88494-9_17.

[9] Christian Szegedy et al. *Intriguing properties of neural networks*. Feb. 19, 2014. DOI: 10.48550/arXiv.1312.6199. arXiv: 1312.6199[cs].

Décider la contextualité de configurations quantiques avec un solveur SAT*

Axel Muller
Université de Franche-Comté, CNRS, institut FEMTO-ST, F-25000 Besançon, France
axel.muller@femto-st.fr

Résumé

Cet article présente mes travaux de recherche en première année de thèse, à l'institut FEMTO-ST à Besançon, sous la direction de M. Alain Giorgetti et le co-encadrement de M. Frédéric Holweck.

Mots-clés : programme quantique, contextualité quantique, vérification formelle

1 Introduction

Mes travaux portent sur la contextualité quantique. Ma thèse est financée par le projet ANR EPiQ (Étude de la Pile Quantique, ANR-22-PETQ-0007) du Plan France 2030 de l'Agence Nationale de la Recherche.

Alain Giorgetti étudie l'informatique quantique depuis 2015 [1]. Il a contribué au projet I-SITE UBFC I-QUINS (*Integrated QUantum Information at the NanoScale*, 01/02/2017-31/01/2020). De 2018 à 2021, il a dirigé la thèse d'Henri De Boutray [2], co-dirigée par Frédéric Holweck et co-encadrée par Pierre-Alain Masson, sur des évaluations calculables de l'intrication et de la contextualité quantiques, dans la perspective de leur vérification formelle [3, 4, 5].

Depuis 2012 Frédéric Holweck mène des recherches sur l'information quantique, en particulier en étudiant les ressources pour les programmes quantiques que sont l'intrication et la contextualité [6, 7, 8]. Il a notamment appliqué ces méthodes à l'algorithme quantique de Grover [9]. Il a porté le projet I-QUINS et co-encadré une thèse sur les algorithmes quantiques, dans le cadre du projet PHYFA (*Photonic plateform for HYperentanglement in Frequency and its Application*), co-porté avec Jean-Marc Merolla (FEMTO-ST).

2 Problématique

La contextualité est un phénomène quantique qui contribue significativement à la supériorité des programmes quantiques, par rapport à leurs homologues classiques. Cependant, il reste encore beaucoup à découvrir, à comprendre et à expliquer sur la manière dont fonctionne cet avantage. La vérification formelle de programmes regroupe diverses techniques d'analyse statique ou dynamique assistées par ordinateur, qui permettent non seulement d'accroitre la confiance dans le logiciel, mais aussi d'expliciter finement et rigoureusement les ressorts internes de son fonctionnement. Ces techniques et les outils correspondants sont progressivement adaptés aux programmes quantiques, mais il reste du travail à faire pour intégrer divers aspects de ce domaine.

Ce sujet de thèse propose de relever le défi de la spécification et de la vérification formelles de propriétés de programmes quantiques, et de l'automatisation de cette dernière, en se concentrant sur les propriétés liées au phénomène de contextualité quantique. Ces programmes et leurs optimisations sont

*Session doctorants

assez complexes pour susciter des interrogations sur leurs bonnes propriétés, dont leur correction, leur complétude et leur complexité algorithmique, qui requièrent des justifications rigoureuses.

Concernant la contextualité, quelques propriétés sont connues, mais elles ont souvent été obtenues soit par des calculs mathématiques manuels, soit par des calculs automatisés pour un petit nombre de bits quantiques. Nous proposons d'assister la vérification de ces propriétés, et la découverte d'autres propriétés analogues, notamment à l'aide de logiciels d'aide à la preuve, tels que Why3 [10] ou Coq [11], afin d'établir ces propriétés avec certitude et quel que soit le nombre de bits quantiques.

Ce défi sera relevé grâce à des adaptations aux propriétés quantiques des techniques de preuve formelle et de test automatique, fondées sur les recherches actuelles en vérification des programmes quantiques. Une attention particulière sera portée sur les caractérisations de la contextualité fondées sur les géométries finies contextuelles [4, 12].

3 Résultats

Mon premier travail publié a porté sur les doilies à N qubits [13]. Pour $N \geq 2$, un doily à N qubits est un doily existant dans l'espace polaire symplectique à N qubits. Ces doilies sont liés aux preuves de contextualité quantique basées sur les opérateurs. En suivant et la stratégie de Saniga et al. [4], qui s'est concentrée exclusivement sur les doilies à trois qubits, nous avons trouvé plusieurs formules donnant le nombre de doilies linéaires et quadratiques pour tout $N > 2$. Ensuite, nous avons conçu un algorithme efficace pour la génération de tous les doilies à N qubits. En utilisant cet algorithme pour $N = 4$ et $N = 5$, nous avons fourni une classification des doilies à N qubits en termes de types d'observables qu'ils présentent et de nombre de lignes négatives dont ils sont dotés. Nous avons énuméré plusieurs résultats remarquables sur les doilies à N qubits, ainsi que quelques caractéristiques spécifiques exhibées par les doilies linéaires. Enfin, nous avons décrit quelques extensions potentielles des doilies linéaires, ainsi que quelques extensions potentielles de notre approche.

Plus récemment, j'ai conçu des algorithmes et un code C pour décider de la contextualité quantique et évaluer le degré de contextualité (une façon de quantifier la contextualité) pour une variété de géométries points-lignes situées dans des espaces polaires symplectiques binaires de petit rang. Grâce à ce code, nous avons pu non seulement reproduire, de manière plus efficace, tous les résultats d'un article récent de de Boutray et al. [12], mais nous avons aussi obtenu un certain nombre de nouveaux résultats [14]. Cet article en cours de révision décrit d'abord les algorithmes, le code C et son interfaçage avec un solveur SAT [15]. Des résultats numériques illustrent ensuite la puissance de cette approche, sur un certain nombre de sous-espaces d'espaces polaires symplectiques dont le rang varie de deux à sept. Les nouveaux résultats principaux sont (i) la non-contextualité des configurations dont les contextes sont des sous-espaces de dimension deux et plus, (ii) la non-existence de sous-espaces négatifs de dimension trois et plus, (iii) des bornes considérablement améliorées pour le degré de contextualité des quadriques elliptiques et hyperboliques pour le rang 4, ainsi que pour une sous-géométrie particulière de l'espace à trois qubits dont les contextes sont les lignes de cet espace, (iv) la preuve de la non-contextualité des perpsets, et enfin et surtout, (v) la nature contextuelle d'une sous-géométrie distinguée d'un doily multi-qubit, appelée *two-spread*, et le calcul de son degré de contextualité.

Il n'est pas aisé d'expliquer à la communauté scientifique des concepts quantiques comme la contextualité, et nos approches formelles pour automatiser sa vérification. C'est pourquoi je propose de présenter, lors des journées 2023 du GDR GPL, en complément de ce résumé, un poster de vulgarisation sur l'utilisation d'un solveur SAT pour détecter la contextualité de configurations quantiques.

Remerciements

Ce projet est soutenu par le projet EPiQ ANR-22-PETQ-0007 du PEPR technologies quantiques dans le cadre du Plan France 2030, et par l'EIPHI Graduate School (contrat ANR-17-EURE-0002).

Références

[1] M. PLANAT et al. "Quantum contextual finite geometries from dessins d'enfants". In : *Int. J. of Geometric Methods in Modern Physics* (avr. 2015). DOI : 10.1142/S021988781550067X.

[2] H. DE BOUTRAY. "Computational studies of entanglement and quantum contextuality properties towards their formal verification". https://hal.archives-ouvertes.fr/tel-03529916. Theses. Université de Franche-Comté, déc. 2021.

[3] H. de BOUTRAY et al. "Mermin polynomials for non-locality and entanglement detection in Grover's algorithm and Quantum Fourier Transform". In : *Quantum Information Processing* 20.3 (mars 2021). ISSN : 1573-1332. DOI : 10.1007/s11128-020-02976-z.

[4] M. SANIGA et al. "Taxonomy of Polar Subspaces of Multi-Qubit Symplectic Polar Spaces of Small Rank". en. In : *Mathematics* 9.18 (sept. 2021). Number: 18 Publisher: Multidisciplinary Digital Publishing Institute, p. 2272. DOI : 10.3390/math9182272.

[5] H. DE BOUTRAY et al. "Automated detection of contextuality proofs with intermediate numbers of observables". In : *18th International Conference on Quantum Physics and Logic (QPL 2021)*. Gdańsk, Poland, juin 2021, p. 3.

[6] F. HOLWECK, J.-G. LUQUE et J.-Y. THIBON. "Geometric descriptions of entangled states by auxiliary varieties". In : *Journal of Mathematical Physics* 53.10 (oct. 2012). Publisher: American Institute of Physics, p. 102203. ISSN : 0022-2488. DOI : 10.1063/1.4753989.

[7] P. LÉVAY et F. HOLWECK. "Embedding qubits into fermionic Fock space: Peculiarities of the four-qubit case". In : *Physical Review D* 91.12 (juin 2015). Publisher: American Physical Society, p. 125029. DOI : 10.1103/PhysRevD.91.125029.

[8] P. LÉVAY, F. HOLWECK et M. SANIGA. "Magic three-qubit Veldkamp line: A finite geometric underpinning for form theories of gravity and black hole entropy". In : *Physical Review D* 96.2 (juill. 2017), p. 026018. DOI : 10.1103/PhysRevD.96.026018.

[9] F. HOLWECK, H. JAFFALI et I. NOUNOUH. "Grover's algorithm and the secant varieties". In : *Quantum Information Processing* 15.11 (nov. 2016), p. 4391-4413. DOI : 10.1007/s11128-016-1445-2.

[10] F. BOBOT et al. *The Why3 Platform*. http://why3.lri.fr/manual.pdf. 2018.

[11] *The Coq Proof Assistant*. http://coq.inria.fr. 1989.

[12] H. DE BOUTRAY et al. "Contextuality degree of quadrics in multi-qubit symplectic polar spaces". In : *Journal of Physics A: Mathematical and Theoretical* 55.47 (nov. 2022), p. 475301. DOI : 10.1088/1751-8121/aca36f.

[13] A. MULLER et al. "Multi-qubit doilies: Enumeration for all ranks and classification for ranks four and five". In : *Journal of Computational Science* 64 (oct. 2022), p. 101853. DOI : 10.1016/j.jocs.2022.101853.

[14] A. MULLER et al. *New and improved bounds on the contextuality degree of multi-qubit configurations*. Mai 2023. DOI : 10.48550/arXiv.2305.10225.

[15] M. S. CHOWDHURY, M. MÜLLER et J. YOU. *A Deep Dive into Conflict Generating Decisions*. en. Mai 2021. DOI : 10.48550/arXiv.2105.04595.