Complétude constructive pour le μ-calcul linéaire

Amina Doumane [1]

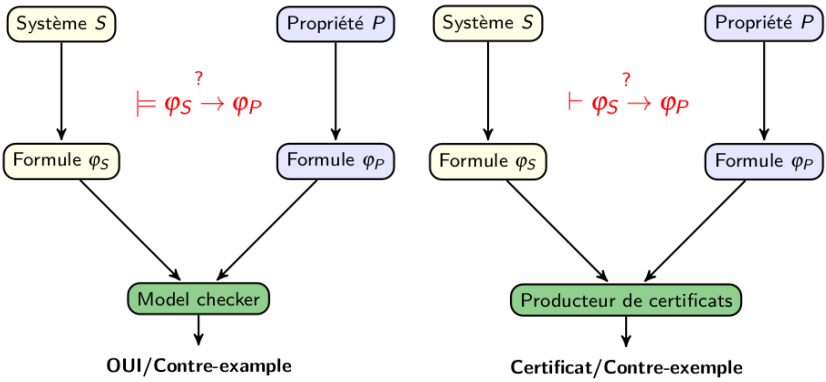
Amina Doumane a soutenu sa thèse [2] *en juin 2017 à l'université Paris-Diderot, thèse préparée à l'Institut de recherche en informatique fondamentale (IRIF) et au Laboratoire spécification et vérification (LSV) sous la direction de David Baelde, Alexis Saurin et Pierre-Louis Curien. Elle effectue actuellement un stage postdoctoral à l'École normale supérieure de Lyon.*

Les programmes informatiques sont maintenant utilisés dans tous les domaines. On les retrouve aussi bien dans des applications critiques (centrales nucléaires, transports en commun, salles des marchés) que dans les utilisations courantes (réseaux sociaux, objets connectés). Cette omniprésence du logiciel rend vitale la problématique de sa sûreté.

Il existe, entre autres, deux paradigmes pour vérifier statiquement que les programmes informatiques répondent aux spécifications pour lesquelles ils ont été conçus. Le premier est le **model-checking** (Figure 1(a)). Il consiste à encoder le système à vérifier par un modèle $\mathscr{M}_S$, la propriété à vérifier par une formule φ_P, et à donner ces deux entrées à un algorithme, un *model-checker*, qui vérifie si le modèle satisfait ou non la formule,

1. https://www.irif.fr/~doumane/

2. « On the infinitary proof theory of logics with fixed points », https://hal.archives-ouvertes.fr/tel-01676953.



(a) Model-checking (b) Vérification déductive

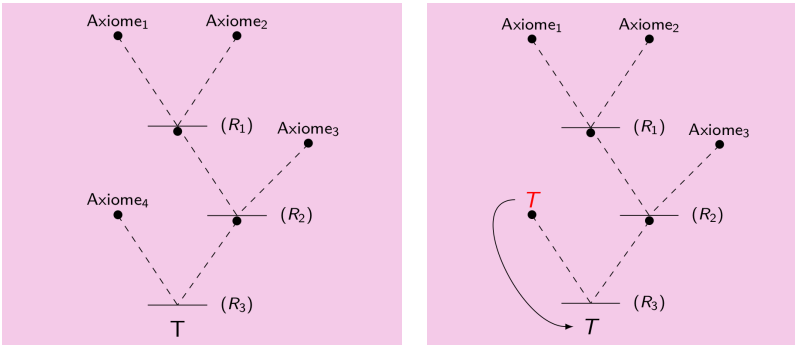
FIGURE 1. Paradigmes du model-checking et de la vérification déductive

ce qui s'écrit $\mathscr{M}_S \models \varphi_P$. Si c'est le cas, cela veut dire que le programme P satisfait effectivement la propriété P. Si ce n'est pas le cas, le *model-checker* fournit un contre-exemple.

Un deuxième paradigme est la **vérification déductive** (Figure 1(b)). Cette technique consiste à modéliser à la fois le système S à vérifier et la propriété P respectivement par des formules logiques φ_P et φ_S. Ensuite on construit une troisième formule $\varphi_S \to \varphi_P$, qui est valide si et seulement si le système S satisfait la propriété P. Cette formule est donnée à un vérificateur qui va soit en construire une preuve si elle est valide, soit retournera un contre-exemple.

L'un des intérêts de cette dernière approche par rapport au *model-checking* est sa capacité à fournir des certificats. En effet, si le *model-checking* produit une réponse booléenne (oui ou non) et un contre-exemple éventuel au problème de la vérification, l'approche déductive produit en plus, dans le cas d'une réponse positive, un certificat, qui est la preuve de $\varphi_P \to \varphi_S$, et qu'on peut communiquer, vérifier indépendamment, générer soit automatiquement, soit avec l'aide d'un assistant de preuve.

Dans l'approche déductive de la vérification, il est nécessaire de savoir construire des preuves de manière automatique. En effet, on aimerait avoir des algorithmes qui prennent en argument une formule (en l'occurrence la formule $\varphi_P \to \varphi_S$) et qui construisent une preuve de cette formule si elle est valide. Je me suis intéressée à de tels algorithmes, pour le μ-calcul linéaire, une logique temporelle avec points fixes utilisée en vérification. Cette logique permet de décrire l'évolution du système dans le temps. Par exemple, on peut écrire en μ-calcul linéaire une formule exprimant la



(a) Preuve usuelle (b) Preuve circulaire

FIGURE 2. Preuve usuelle vs preuve circulaire

spécification « le système va finir par atteindre un état satisfaisant la propriété φ » ou encore « le système satisfait à tout moment la propriété φ ».

Dans son article fondateur, Kozen [6] a proposé un système de preuve pour le μ-calcul. Treize ans plus tard, Kaivola [5] a montré sa **complétude**, en d'autres termes il a montré que toute formule valide est prouvable. Cette preuve de complétude est basée sur des arguments complexes et non constructifs, qui ne donnent pas de moyen simple de construire une preuve pour une formule valide. Or c'est ce dont on a besoin pour faire de la vérification déductive.

Mon travail a été de proposer une nouvelle preuve de complétude pour le μ-calcul linéaire qui est **constructive**, c'est-à-dire qui construit une preuve pour toute formule valide [3, 1, 2]. Pour cela j'ai identifié trois difficultés lorsqu'il s'agit de prouver les formules du μ-calcul. J'ai remarqué de plus que ces difficultés existent déjà en théorie des automates sous des noms différents. Heureusement, en théorie des automates, ces difficultés sont toutes simplifiables, et il existe des algorithmes efficaces pour les éliminer. Mon idée a été donc d'importer ces algorithmes de la théorie des automates vers la théorie de la démonstration, en me basant sur la très célèbre correspondance entre le μ-calcul et la théorie des automates [4].

Pour arriver à implémenter ces algorithmes je suis passée par les **preuves circulaires**. Les preuves que l'on utilise usuellement, en mathématique et en informatique, ont la forme d'arbres finis, comme celui de la Figure 2(a). Les feuilles de ces arbres sont des axiomes ($axiome_1$, $axiome_2$, ...), qui sont des énoncés admis comme triviaux, qu'on va combiner en utilisant des règles de déduction R_1, R_2, ... pour construire des énoncés de plus en plus complexes jusqu'à arriver à l'énoncé du théorème T qu'on vise à prouver.

Les preuves circulaires, elles, ont la particularité de nous permettre d'utiliser l'énoncé qu'on cible comme s'il était un axiome (Figure 2(b)). Cette utilisation de l'énoncé comme axiome est souvent dénotée par un arc, qui crée un cycle dans la preuve, d'où le nom preuves circulaires. Bien évidemment, si on ne contraint pas l'utilisation de ces cycles on peut déduire des contradictions, d'où la nécessité d'imposer des conditions, qu'on appelle conditions de validité, pour éviter les cercles vicieux.

Pour revenir à ma preuve de complétude, lorsque l'on a une formule valide et qu'on lui cherche une preuve dans le système de Kozen, on commence par lui trouver une preuve circulaire, en implémentant les algorithmes venant de la théorie des automates. Ensuite, on transforme cette preuve circulaire en une preuve de Kozen, en utilisant un algorithme que j'ai développé dans ma thèse.

Pour conclure, ce résultat est une belle preuve que l'interdisciplinarité peut être très fructueuse, et que souvent nos problèmes sont résolus chez notre voisin qui travaille dans un domaine différent. Il suffit parfois d'avoir le courage d'aller lui parler...

Références

[1] Amina Doumane. Constructive completeness for the linear-time μ-calculus. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.

[2] Amina Doumane. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. PhD thesis, Paris Diderot University, France, 2017.

[3] Amina Doumane, David Baelde, Lucca Hirschi, and Alexis Saurin. Towards completeness via proof search in the linear time μ-calculus : The case of büchi inclusions. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16, New York, NY, USA, July 5-8, 2016*, pages 377–386, 2016.

[4] David Janin and Igor Walukiewicz. Automata for the modal mu-calculus and related results. In Jirí Wiedermann and Petr Hájek, editors, *MFCS '95*, volume 969 of *LNCS*, pages 552–562. Springer, 1995.

[5] Roope Kaivola. Axiomatising linear time mu-calculus. In *CONCUR '95 : Concurrency Theory, 6th International Conference, Philadelphia, PA, USA, August 21-24, 1995, Proceedings*, pages 423–437, 1995.

[6] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.