Goéland, prouveur concurrent basé sur la méthode des tableaux

16 Avril 2023 Matthieu Pierret Montpellier, France

L'utilisation de la logique mathématique pour la vérification formelle de programmes informatiques est une pratique de plus en plus courante pour garantir la fiabilité et la sécurité de ceux-ci. Le prouveur concurrent Goéland est un outil basé sur la méthode des tableaux, qui a suscité un intérêt croissant ces dernières années. Dans le cadre d'une étude bibliographique, nous allons présenter une synthèse des travaux actuels sur le prouveur Goéland, en mettant l'accent sur ses avantages, ses limites et ses perspectives d'avenir. Nous examinerons également les problématiques liées à l'utilisation de la logique mathématique pour la vérification formelle de programmes, en nous concentrant sur les aspects les plus pertinents pour le sujet de notre étude. Notre objectif est de fournir une analyse complète et à jour des connaissances actuelles sur le prouveur concurrent Goéland et certains autres. Ce genre d'outil est très utilisé pour garantir la sécurité et la fiabilité des systèmes de traitements ou de communication, en particulier dans des domaines tels que la finance, la santé, la défense ou l'astronautique, où les erreurs peuvent avoir de graves conséquences.

1 Principe de vérification par tableaux

Comme dit précédemment, Goéland utilise la méthode des tableaux pour ses vérifications, cette dernière est très semblable à la méthode des séquents mais appliquée à la logique du premier ordre. Le calcul des séquent consiste à prouver par déduction via les règles de calcul que la négation de la formule que l'on veut démontrer est insatisfiable, c'est à dire qu'il n'existe pas de modèle pour que cette formule soit vraie. La méthode des tableaux est donc une méthode de raisonnement logique qui permet de prouver ou de réfuter une formule logique en utilisant des règles d'inférence et des techniques de simplification. Elle repose sur la construction d'un tableau où chaque ligne représente une valuation des variables propositionnelles de la formule à prouver ou à réfuter. L'objectif de la méthode est de déterminer si la formule est vraie ou fausse en examinant toutes les valuations possibles. Pour ce faire, la méthode utilise des règles d'expansion qui permettent d'ajouter de nouvelles lignes au tableau en fonction des règles syntaxiques de la logique du premier ordre. Il existe également des règles de réduction qui permettent de simplifier le tableau en éliminant les lignes qui sont équivalentes à d'autres lignes déjà présentes. En fin de compte, si toutes les valuations possibles ont été examinées et qu'aucune contradiction n'a été trouvée, la formule est considérée comme vraie. Si, en revanche, une contradiction est apparue, la formule est considérée comme fausse. Pour résumer : la méthode des tableaux est un outil puissant pour la vérification formelle des systèmes informatiques, car elle permet de prouver de manière formelle la validité ou l'invalidité des formules logiques parfois complexes.

La méthode des tableau repose sur des règles simples, de manière similaire aux séquents mais avec des sous formules. Voici les règles de cette méthode, elle seront utiles dans la compréhension de l'exemple ci-dessous : [3] :

- Règles de cloture $\odot$ lorsque l'on trouve un littéral et son opposé, ou en la présence de $\top$ ou $\bot$.
- Règles de conjonctions α qui ne forment pas de nouvelle branche et qui permettent de casser les connecteurs.
- Règles de disjonction β qui elles forment de nouvelles branches contenant une partie de la formule d'origine.
- Règles existentielles δ qui sont les règles de skolémisation qui introduisent un nouveau symbole, dit *symbole de Skolem*.
- Règles universelles γ qui cassent les quantifica-

teurs existentiels pour créer des métavariables, ou variables libres.

Le but de la méthode est évidemment de prouver qu'une formule est valide. Pour ce faire, la formule initiale est niée, et la méthode consiste à prouver que sa négation est insatisfiable. En utilisant cette méthode, on trouve qu'une formule est insatisfiable si toutes ses branches sont closes (se terminent par une des règles de cloture $\odot$).

$$\dfrac{\dfrac{\dfrac{\dfrac{p(a) \wedge \neg p(b) \wedge \forall x\ (p(x) \Leftrightarrow (\forall y\ p(y)))}{p(a), \neg p(b), \forall x\ (p(x) \Leftrightarrow (\forall y\ p(y)))}\,\alpha\wedge}{p(X) \Leftrightarrow (\forall y\ p(y))}\,\gamma\forall}{\dfrac{p(X/a), \forall y\ p(y)}{p(Y/b)}\,\odot}\,\gamma\forall \quad \dfrac{\neg p(X/a), \neg(\forall y\ p(y))}{\odot}\,\odot}\,\beta\Leftrightarrow$$

La vérification se lit du haut vers le bas, avec à chaque étape, la règle indiquée à droite. Afin de mettre l'accent sur la méthode, à chaque étape les formule qui n'ont pas été touchées ne sont pas réécrites.

Dans cet exemple, nous appliquons d'abord la règle $\alpha\wedge$ deux fois, sur les connecteurs racine. On peut ensuite y appliquer $\gamma\forall$ pour obtenir une variable libre X et $(\forall y\ p(y))$. On a maintenant l'application de $\beta \leftrightarrow$ qui créée deux sous formules : on dit qu'elle *branche*. Sur la première sous formule nous pouvons réappliquer $\gamma\forall$ pour obtenir la variable libre Y à laquelle nous assignerons la valeur b, afin d'avoir $\neg p(b)$ (que nous avons obtenu à la seconde étape) et $p(b)$ sur la même ligne, ce qui conduira à la clôture de celle-ci via $\odot$. Sur la seconde branche, nous assignons X à $p(a)$ afin d'avoir $p(a)$ (lui aussi obtenu plus tôt) et $\neg p(a)$ et la clôturer via $\odot$. A noter que nous n'avons pas instancié X à $\neg a$ sur la première branche pour pouvoir le garder sur la seconde, et éviter un choix injuste de la variable libre, qui aurait conduit à ne pas réussir à prouver la formule [4].

Un des argument qui à poussé à l'utilisation de la méthode des tableaux par Goéland est sa capacité à pouvoir être utilisée sur des formules vierges et donc non skolémisées. Pour la logique du premier ordre, les formules universelles sont usuellement gérées par les règles universelles γ avec la création de variables libres [4]. Cependant, cette méthode est dite destructrice, car les branches encore ouvertes peuvent partager des variables libres et fausser l'évaluation de la formule dans d'autres branches (comme dans l'explication ci-dessus). Reprendre l'évaluation avec des variables libres différentes n'est pas une solution, car cette nouvelle évaluation créera un nouveau *symbole de Skolem* qui engendrera le même problème. Cela signifie que dans certaines formules, une branche qui sera explorée avant une autre pourrait fausser l'évaluation des suivantes.

2 La concurrence pour la vérification

C'est pour palier ce problème que la concurrence est utilisée par Goéland. En effet, la concurrence permet l'exploration en simultané de plusieurs branches et ainsi détecter les problèmes liés aux variables libres. De plus, la concurrence permet d'exploiter l'exécution parallèle et donc le multithreading des processeurs récents afin d'incrémenter l'efficacié du processus. Combiner ce principe avec les variable libres nous donne une ce que l'on appelle la méthode des tableaux avec variables libres. Goéland est donc implémenté avec le langage de programmation Go, puisqu'il supporte nativement la concurrence [6]. Goéland est également équipé d'un système de déduction modulo, très utile que ce soit pour la recherche de preuve en automatique, ou avec des règles de réécriture ajoutées maanuellement (très utilisé dans Goéland) [2]. La déduction modulo est une pratique qui permet d'éviter les difficultés liées à la manipulation d'équations complexes pour se concentrer sur l'aspect logique et formel du raisonnement [5]. En effet, l'utilisation de goroutines permet à Goéland de stocker les données des métavariables, avec une limite de réintroduction de symbole qui est choisie par l'utilisateur.

3 Résultats : comparaisons et conclusions

Le prouveur Goéland a été testé sur deux catégories de problèmes : des problèmes syntaxiques dépourvus d'égalités (SYN) et des problèmes sur la théorie des ensembles. Il a aussi été comparé sur les résultats de ses différents tests avec trois autres prouveurs basés sur la méthode des tableaux : *Zenon* (bien qu'il n'ait pas été initialement développé pour des équations sans

égalités [1]), *Princess* et *LeoIII* ; ainsi que deux prouveurs par saturation : *Vampire* et *E*. Les conditions de tests étaient les suivantes : processeur **Intel Xeon E5-2680 v4 2.4GHz 2×14-core** avec 128 GB de mémoire, avec pour chaque preuve un temps maximum de 300 secondes [4]. Dans les résultats suivants, sera cité Goéland+DMT, DMT signifiant Deduction Modulo Theory. Le principe de base de la déduction modulo est d'utiliser des fonction de réécriture sur des formules en utilisant des équivalences logiques. Cela permet parfois d'alléger les formules ou de faciliter la résolution en diminuant le nombre de branches. Voici un tableau montrant les résultats obtenus[4] :

	SYN (263 problems)		SET (464 problems)	
Goéland	**199** (190 s)		**150** (4659 s)	
Goéland+DMT	**199** (196 s)	(+0, −0)	**278** (1292 s)	(+142, −14)
Zenon	**256** (67 s)	(+60, −3)	**150** (562 s)	(+75, −75)
Princess	**195** (189 s)	(+1, −5)	**258** (1168 s)	(+141, −33)
LeoIII	**195** (268 s)	(+1, −5)	**177** (2925 s)	(+77, −50)
E	**261** (168 s)	(+62, −0)	**363** (2377 s)	(+223, −10)
Vampire	**262** (13 s)	(+63, −0)	**321** (4122 s)	(+188, −17)

Les chiffres entre paranthèses sont le nombre de résolutions relatives à Goéland, (+15, -2) signifie que le prouveur a résolu 15 problèmes que Goéland n'a pas résolu, et inversement, qu'il a résolu 2 problèmes de moins.

Les temps et résultats de Goéland par rapport aux trois autres prouveurs basés sur la méthode des tableaux sont identique, et même un petit peu meilleurs dans certains cas. Les résultats de Goéland avec ou sans DMT sont assez similaire, bien que pour les problèmes SET l'ajout du DMT s'avère être très efficace. Il ne faut cependant pas oublier que Goéland est encore en début de développement, et que beaucoup de fonctionnalités sont déjà prévues pour y être implémentées, comme le polymorphisme et les raisonnements arithmétiques. Il est d'ailleurs prévu d'intégrer un mode de vérification assistée, où la machine peut montrer pas à pas à l'utilisateur comment arriver à une preuve complète. Il faut également souligner que plusieurs logiciels dans le monde ont fait appel aux vérifications formelles (comme CompCert C, Xmonad window manager ou sel4 par exemple). Il est par conséquent à conclure que ce genre de prouveur est un sujet de recherche très prometteur, que ce soit par la nécessité de sécurité dans certaines applications concrètes, ou pour faire avancer d'autres pans de la recherche. Les travaux concernant les prouveurs logique sont encore aujourd'hui chose courante, ce qui laisse entrevoir d'éventuelles découvertes à venir.

Références

[1] Richard Bonichon, David Delahaye, and Damien Doligez. Zenon : an Extensible Automated Theorem Prover Producing Checkable Proofs. In *LPAR 2007 - 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4790 of *Lecture Notes in Computer Science*, pages 151–165, Yerevan, Armenia, October 2007. Springer.

[2] Guillaume Burel, Guillaume Bury, Raphaël Cauderlier, David Delahaye, Pierre Halmagrand, and Olivier Hermant. First-order automated reasoning with theories : When deduction modulo theory meets practice. *Journal of Automated Reasoning*, 64, 08 2020.

[3] Julie Cailler. Tableaux : règles et exemples. Private course, 2023.

[4] Julie Cailler, Johann Rosain, David Delahaye, Simon Robillard, and Hinde Lilia Bouziane. Goéland : A concurrent tableau-based theorem prover (system description). In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning*, pages 359–368, Cham, 2022. Springer International Publishing.

[5] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning*, 31(1) :33–72, 2003. Article dans revue scientifique avec comité de lecture. internationale.

[6] Mihalis Tsoukalos. *Mastering Go : Create Golang production applications using network libraries, concurrency, machine learning, and advanced data structures*. Packt Publishing Ltd, 2019.