Sujet de stage M2RI

Bounded UDPOR

Martin Quinson, Thierry Jéron

Equipes Myriad et Sumo, IRISA-INRIA, Rennes

https://team.inria.fr/myriads/, http://www.irisa.fr/sumo/

Mots-clé: systèmes distribués, vérification, ordres partiels, causalité, concurrence,

Description

***Contexte*:**

Le model checking est une méthode formelle qui consiste à vérifier des propriétés logiques d'un modèle de système par exploration exhaustive de ses comportements. Nous nous focalisons ici sur la vérification de systèmes distribués asynchrones et sur les techniques où le modèle est construit par l'exécution (réelle ou simulée) du système. Le model checking est simple d'utilisation par des non-spécialiste, mais initialement limité à des systèmes de taille modeste à cause de l'explosion combinatoire due à l'exploration exhaustive de l'espace d'états. Depuis plusieurs décennies les recherches s'emploient donc à trouver des techniques alternatives permettant le passage à l'échelle (e.g. en exploitant des propriétés de symétrie, des codage efficaces, etc).

Parmi ces techniques, la réduction par ordre partiel (POR pour *Partial-Order Reduction*) tire parti de l'indépendance causale entre événements, identifiée par la relation happened-before (cf. https://en.wikipedia.org/wiki/Happened-before). Le principe est d'explorer au moins, et si possible une seule linéarisation par *trace de Mazurkiewicz* (classe de comportements équivalents à commutation près des événements indépendants adjacents), pour des classes de propriétés insensibles à ces commutations. Les méthodes proposées diffèrent alors sur la définition, dans chaque état exploré, d'un sous-ensemble des transitions suffisantes pour garantir la couverture de toutes les traces de Mazurkiewicz et sur le compromis à déterminer entre réduction du nombre de traces et précision d'analyse pour calculer cet ensemble. En amont, une des difficultés est d'affiner la relation d'indépendance entre événements du système. Pour ce faire, les algorithmes de la famille DPOR (*Dynamic Partial Order Reduction*) utilisent des informations dynamiques, i.e. connues à l'exécution/exploration du système (e.g. la valeur des variables impliquées).

D'autres alternatives issues de la théorie de la concurrence ont été aussi étudiées afin de palier aussi à l'explosion combinatoire, comme celle des structures d'événements et leurs dépliages (*unfoldings*). Ceux-ci permettent de représenter l'ensemble des comportements du système de façon exponentiellement plus compacte que l'espace d'état, en explicitant les liens de causalité, d'indépendance et de conflit entre événements. Malheureusement, construire ce dépliage pour un système donné est un

problème NP-difficile dans le cas général.

En 2015, les auteurs de [1] ont proposé de combiner dépliages et DPOR dans l'algorithme UDPOR (pour *Unfolding based Dynamic Partial Order Reduction*) pour la classe de programmes multithreadés synchronisés par des mutex. Dans ce contexte, UDPOR permet de construire le dépliage dynamiquement en temps polynomial, et en l'utilisant pour sélectionner les sous-ensembles de transitions suffisant à explorer dans DPOR. Une extension a ensuite été proposée [2]. Dans [3], nous avons adapté UDPOR à un modèle de programmation de systèmes distribués asynchrones interagissant par échanges de messages et par mutex, classe de programmes suffisamment vaste pour couvrir la majorité des programmes de calcul intensif utilisés de nos jours (cf. https://en.wikipedia.org/wiki/Supercomputer).

Par ailleurs, les auteurs de [4] ont proposé une adaptation orthogonale de DPOR en limitant l'exploration de l'espace d'états à une profondeur bornée. La difficulté est alors de s'assurer que tout comportement qui satisfait la limite de profondeur soit représenté, et ne soit donc pas indument coupée par équivalence d'un préfixe à un autre comportement.

***Sujet*:**

L'objectif du stage est d'étudier des extensions possibles à ces travaux, dans le contexte de la vérification de programmes distribués asynchrones. Suivant les aspirations du candida, plusieurs pistes de travail sont envisageables: élargissement des modèles de programmation considérés, précisions des analyses, des relations d'indépendance, ou améliorations algorithmiques.

Une première piste serait d'élargir la classe de programmes à laquelle s'applique notre adaptation d'UDPOR. Le modèle de programmation pourrait être enrichi de mécanismes de synchronisation comme les sémaphores ou les barrières, ou de mécanismes de communications comme les *one-sided communications* de MPI.
On pourra aussi affiner les relations d'indépendances proposées pour notre modèle, certaines étant des approximations, certes simples à calculer, mais trop imprécises.
À terme, il serait intéressant d'établir une variante de l'algorithme d'UDPOR préservant la complétude à profondeur bornée, voire de généraliser à des conditions d'arrêt d'exploration qui permettraient à l'utilisateur de concentrer la recherche sur des bugs complexes mais plus probables.

D'autres perspectives à plus long terme peuvent être envisagées:

Notre modèle de programmation pourrait intégrer les pannes et *reboot* partiels du système distribué. Sachant qu'aucun système ne peut respecter des propriétés de bon fonctionnement quand l'intégralité de ses composants sont systématiquement en panne, on pourrait borner le nombre de pannes constatées sur chaque trace d'exécution.

Les travaux cités jusque là se limitent aux propriétés de sûreté (un événement indésirable n'arrive jamais, un peu comme un *assert* en programmation). Il serait intéressant d'adapter UDPOR aux propriétés de vivacité (un événement désiré finira par arriver), ce qui revient à vérifier l'absence de boucles infinies indésirables dans l'espace d'états. Ceci constitue un défi ambitieux car il s'agit de préserver l'information des cycles dans les dépliages construits.

Notons enfin que notre adaptation d'UDPOR est actuellement en cours d'implémentation effective dans l'outil d'évaluation pour systèmes distribués SimGrid (co-développé dans l'équipe Myriads). L'objectif à terme est de permettre la vérification de vrais systèmes distribués de calcul à haute performance grâce aux meilleurs algorithmes de model checking. Une interaction avec l'ingénieur en charge de cette tâche pourra être envisagée pour comprendre les enjeux applicatifs.

Prérequis : le stage s'adresse à un étudiant attiré par les méthodes formelles, les modèles sous-jacents, en particulier les modèles de programmation distribuées, les concepts de la concurrence, etc.

———————————————————————————————

Keywords: distributed system, verification, partial orders, causality, concurrency,

Context: Model checking is a formal method that consists in verifying logical properties of a system model by exhaustive exploration of its behavior. We focus here on the verification of asynchronous distributed systems and on techniques where the model is built by the execution (real or simulated) of the system. Model checking is easy to use by non-specialists, but initially limited to systems of modest size because of the combinatorial explosion due to the exhaustive exploration of the state space. Since several decades, research has been focused on finding alternative techniques allowing scaling (e.g. by exploiting symmetry properties, efficient coding, etc.).

Among these techniques, Partial-Order Reduction (POR) takes advantage of the causal independence between events, identified by the happened-before relation (see https://en.wikipedia.org/wiki/Happened-before). The principle is to explore at least, and if possible only one linearization by Mazurkiewicz trace (class of equivalent behaviors by commutation of adjacent independent events), for classes of properties insensitive to these commutations. The proposed methods then differ on the definition, in each explored state, of a subset of the transitions sufficient to guarantee the coverage of all the Mazurkiewicz traces, and on the trade-off to be found between the reduction of the number of traces and the accuracy of the analysis to compute this set. Beforehand, one of the difficulties is to refine the independence relation between events in the system. To do this, the algorithms of the DPOR (Dynamic Partial Order Reduction) family use dynamic information, i.e. information that is known at execution/exploration time (e.g. the values of the variables involved).

Other alternatives stemming from concurrency theory have also been studied in order to overcome the combinatorial explosion, such as event structures and their unfoldings. These allow to represent all the behaviors of the system in an exponentially more compact manner than the state space, by making explicit the relations of causality, independence and conflict between events. Unfortunately, constructing this unfolding for a given system is an NP-hard problem in the general case.

In 2015, the authors of [1] proposed to combine unfolding and DPOR in the UDPOR (for Unfolding based Dynamic Partial Order Reduction) algorithm for the class of multithreaded programs synchronized by mutexes. In this context, UDPOR allows us to build the unfolding dynamically in polynomial time, and then use it to select

sufficient subsets of transitions to explore in DPOR. An extension is proposed in [2]. In [3], we then adapted UDPOR to a programming model of asynchronous distributed systems interacting through message exchanges and mutexes, a class of programs large enough to cover the majority of HPC programs in use today (see https://en.wikipedia.org/wiki/Supercomputer).

Moreover, the authors of [4] proposed an orthogonal adaptation of DPOR by limiting the exploration of the state space to a bounded depth. The difficulty is then to ensure that any behavior that satisfies the depth limit is represented, and is therefore not unduly cut off by equivalence of a prefix to another behavior.

Subject:

The objective of the internship is to study possible extensions to this work, in the context of the verification of asynchronous distributed programs. Depending on the aspirations of the candidate, several avenues of work are possible: extension of the considered programming models, precision of the analyses, of the independence relations, or algorithmic improvements.
A first direction would be to enlarge the class of programs to which our adaptation of UDPOR applies. The programming model could be enriched with synchronization mechanisms such as semaphores or barriers, or with communication mechanisms such as MPI's one-sided communications.
We could also refine the independence relations proposed for our model, some of them being approximations, certainly simple to calculate, but too imprecise.
Eventually, it would be interesting to establish a variant of the UDPOR algorithm that preserves completeness at bounded depths, or even to generalize to conditions for halting exploration that would allow the user to concentrate the search on complex but more likely bugs.

Other longer term perspectives can be envisaged:
Our programming model could incorporate partial failures and reboots of the distributed system. Knowing that no system can respect the properties of correct functioning when all its components are systematically faulty, we could limit the number of failures observed on each execution trace.

The works quoted so far are limited to safety properties (an unexpected) event never happens, a bit like an assert in programming). It would be interesting to adapt UDPOR to liveness properties (a desired event will eventually happen), which means checking the absence of unwanted infinite loops in the state space. This is an ambitious challenge since it is necessary to preserve the information of the cycles in the constructed unfoldings.

Finally, our adaptation of UDPOR is currently being implemented in the evaluation tool for distributed systems SimGrid (co-developed in the Myriads team). The ultimate goal is to allow the verification of real high performance distributed computing systems using the best model checking algorithms. An interaction with the engineer in charge of this task could be envisaged to understand the application issues.

Skills: the internship is intended for students interested in formal methods, the

underlying models, in particular distributed programming models, concepts of concurrency, etc.

Bibliographie

[1] Rodriguez, Sousa, Sharma, Kroening.
Unfolding-based partial order reduction.
26th International Conference on Concurrency Theory (CONCUR'15).
https://arxiv.org/pdf/1507.00980.pdf

[2] Nguyen, Rodriguez, Sousa, Coti, Petrucci: Quasi-optimal Partial Order Reduction.
30th International Conference on Computer Aided Verification, CAV'18.
https://doi.org/10.1007/ 978-3-319-96142-2_22

[3] Pham, Jéron, Quinson.
Unfolding-based Dynamic Partial Order Reduction of Asynchronous Distributed Programs.
39th International Conference on FormalTechniques for Distributed Objects, Components, and Systems
(FORTE 2019).
https://hal.inria.fr/hal-02109769/document

[4] Coons, Musuvathi, S. McKinley.
Bounded Partial-Order Reduction.
Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming
Systems Languages & Applications (OOPSLA'13).
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bpor-oopsla-2013.pdf

Contacts

Martin Quinson
Email : Martin.Quinson@irisa.fr
https://people.irisa.fr/Martin.Quinson/
Tél : (+33/0)6 19 31 06 92

Thierry Jéron
Email : Thierry.Jeron@inria.fr
http://www.irisa.fr/prive/jeron/
Tél : (+33)6 07 14 24 61