Equivalence checking for asynchronous software

Author(s):
Haroud, Malek

Publication Date:
2005

Permanent Link:

https://doi.org/10.3929/ethz-a-005124898 →

Diss. ETH No. 16312

Equivalence Checking for Asynchronous Software

A dissertation submitted to the
Swiss Federal Institute of Technology Zürich
ETH Zürich

for the degree of
Doctor of Technical Sciences

presented by
Malek Haroud
Dipl. Elektrotek-Ing. EPFL
born July 20, 1972
citizen of Plaffein FR,
Switzerland

accepted on the recommendation of
Prof. Dr. Armin Biere, examiner
Prof. Dr. Lothar Thiele, co-examiner

October 2005

Abstract

Model-driven engineering (MDE) [59] aims at describing and implementing complex systems starting from abstract models as opposed to the traditional code-centric methods. It is based on the idea of platform independent models and platform description models that can be realized using a variety of middleware and programming languages into platform specific models. Therefore, the MDE paradigm is more adapted than traditional approaches to cope with today's systems complexity.

The specification and description language SDL belongs to the model-centric paradigm and has been successfully used at the telecommunication standardization bodies and in the industry as well. However, if we compare it to the code-centric approach, the time and effort invested in developing a platform independent model needs to be compensated. Consequently, model-to-code transformers are provided to derive automatically the implementation from the models. Ensuring the correctness and the reliability of these transformers is the key point that is addressed in this work.

This thesis addresses the problem of equivalence checking in the context of asynchronous software from a practitioner point of view. Our research builds on previous and ongoing work on compiler verification and translation validation. But, rather than simply focusing on the synchronous or purely sequential programs, it addresses the parallel asynchronous composition aspects as well.

In this work, we provide a verification methodology in which developers can automatically verify the refinement of an abstract reference model into the final target code. In such a flow, there is absolutely no need to re-validate again the target code obtained manually or by using a compiler. Without loss of generality, we treated the case where SDL is the source language and C the target implementation language.

In order to enable equivalence checking, both abstract and concrete programs need first to be translated into an intermediate form. The design of this intermediate representation was an important challenge of this thesis. Following the principle of union of concepts, we proposed an intermediate representation which coerces relevant aspects of both languages. This intermediate representation was defined in terms of its syntax and operational semantics.

Once both programs are translated into the intermediate representation, correspondence links are setup between the abstract and the concrete programs thanks to the concept of *cutpoints*. Their usage is a key concept in our method since they allow to subdivide the verification problem into smaller sub-problems. Cutpoints are also used to break the loops as in Floyd's method [57].

Under such correspondence, we formalized the notion of equivalence by providing an inductive argument based on bisimulation. The sequential case was addressed together with two cases of the asynchronous parallel composition namely the server model and the activity thread model.

Our equivalence checking method was motivated by pragmatics. We insisted therefore to achieve a fully automatic method to check the equivalence systematically at each run of the compiler. Since program equivalence is undecidable for most interesting cases, we have imposed several simplifying assumptions under which the equivalence checking becomes algorithmically solvable and scalable as well. The main assumptions that underly this research concern the restriction to a subset of SDL, the compilation scheme and the scheduling regime.

To demonstrate the effectiveness of our approach, we have built a tool called SCEC. This tool is written in ANSI C. Flex and Bison specifications were also used to produce the scanners and the parsers for C and SDL. A WxWidgets based graphical user interface has been developed in order to browse the intermediate representation of both programs. Starting from the syntax tree, SCEC can record all the transformations that are applied on the trees up to the final normal form. Proof-obligations are generated automatically and are discharged using the ICS tool (*i.e.; Integrated Canonizer and Solver ICS*[30]). Currently, SCEC handles the Telelogic CAdvanced compiler only. Rewrite rules are hard-coded inside the tool. We claim that using a term rewriting engine [82], other code generators can be easily retargeted.

The experiments part contains two case studies: the IEEE 802.11 MAC layer and the MIPv6 protocol. The first case study represents a typical industrial software project. SCEC verified the C source code against its associated SDL specification in about one minute. In addition, these two case studies showed that the selected SDL sub-set is sufficiently expressive to describe real-world protocols. One major outcome is that during the verification process of the MAC layer, a subtle bug in the CAdvanced compiler was discovered. This result is very encouraging and shows that the development of such equivalence checker does pay off. We believe that the equivalence checking concept can be generalized to *Domain-specific languages* whose user base of their compilers are necessarily smaller and therefore less robust in comparison to main stream compilers.

Résumé

L'*ingénierie dirigée par les modèles* (IDM) [59] a pour objectif de décrire et de mettre en oeuvre des systèmes complexes en partant de modèles abstraits par opposition à l'approche plus traditionnelle ciblant directement un code proche de la réalisation finale. L'idée de base repose sur une séparation entre des modèles indépendants des plateformes cibles et de ceux présents dans les produits finaux. Par conséquence, le paradigme IDM permet une meilleure maîtrise de la complexité des systèmes actuels que l'approche traditionnelle.

Le langage de description et de spécification LDS suit le concept IDM et a été intégré avec succès aussi bien par les organismes de standardisation que par l'industrie. L'effort de développement d'un modèle dans un langage comme LDS ne pourra se compenser que par l'utilisation de compilateurs dédiés permettant d'en dériver une réalisation efficace. La problématique centrale de cette thèse est de garantir la justesse de la transformation du modèle abstrait vers sa réalisation sur cible. En d'autres termes, il s'agit de s'assurer que le compilateur n'a pas introduit d'erreurs. Cette thèse traite du problème de la vérification d'équivalence dans le contexte de logiciels asynchrones sur un plan pratique. Notre recherche repose sur les travaux liés à la vérification de compilateurs mais ne se focalise par seulement sur des programmes synchrones ou séquentiels mais s'intéresse également à la composition parallèle et asynchrone.

Ce travail met à disposition une méthodologie de vérification qui permet aux développeurs de vérifier automatiquement la justesse du raffinement du modèle abstrait vers le code cible. Sans restriction du champ d'application, nous nous sommes intéressés au cas où le programme source est exprimé en LDS et traduit en code ANSI C. Afin de permettre la vérification d'équivalence, le modèle abstrait ainsi que le code cible correspondant doivent être exprimé dans une représentation intermédiaire commune. La conception d'une telle représentation a été le défit majeur relevé par ce travail. Nous avons opté pour la définition d'un langage intermédiaire qui unifie les types d'énoncés de base des deux langages LDS et ANSI C. Ce langage intermédiaire a été formellement exprimé tant du point de vue syntaxique que de sa sémantique opérationelle.

Une fois que les deux programmes LDS et C sont traduits vers la représentation intermédiaire, des points de correspondance sont introduits grâce au principe des points de coupures. Leur utilisation permet de décomposer le problème de vérification en un ensemble de sous-problèmes plus gérables. Les points de coupures sont également utilisés

pour ouvrir les boucles d'itération comme dans la méthode de Floyd [57]. Par ailleurs, nous avons formalisé la notion d'équivalence en utilisant un argument inductif basé sur la relation de bisimulation. Le cas séquentiel a été traité ainsi que deux cas de la composition parallèle asynchrone à savoir le modèle du serveur et celui de la sérialisation des tâches.

Notre méthode de vérification d'équivalence répond à un problème concret auquel fait face l'industrie. Nous avons donc insisté sur la réalisation d'un logiciel qui soit complètement automatique et qui permet de vérifier chaque compilation systématiquement. Bien entendu, selon la théorie de la calculabilité, il n'existe aucun algorithme pouvant d'une manière générale prouver l'équivalence de deux programmes. Néanmoins, nous nous sommes attelés à trouver les hypothèses supplémentaires qui permettent de fournir un algorithme de vérification qui puisse traîter des problèmes de taille industrielle. Ces hypothèses font partie de trois catégories: la restriction du langage LDS à un sous-ensemble, le modèle de compilation ainsi que la méthode d'ordonnancement des processus LDS.

Un prototype appelé SCEC a été développé dans le cadre de cette thèse. Ce prototype est écrit en ANSI C et se base sur des spécifications Flex et Bison pour l'analyse lexicale et syntaxique des deux langages LDS et ANSI C. Une interface graphique permet de visualiser toutes les étapes intermédiaires de transformation des deux programmes vers la forme commune. Les obligations de preuves sont levées grâce à ICS [30]. Actuellement, SCEC vérifie les compilations réalisées par CAdvanced de Telelogic. Les règles de réécritures sont pour l'instant figées mais on pourra envisager dans le futur, l'utilisation d'un interpréteur de règles de réécriture [82] afin de cibler d'une manière plus flexible d'autres compilateurs.

La partie expérimentale comprend deux études de cas: la couche MAC 802.11 ainsi que le protocole MIPv6. Le premier cas, représente une situation typique d'un projet industriel. SCEC a pu vérifier la totalité du LDS et du C en moins d'une minute. De surcroît, ces deux études de cas montrent bien que le sous-ensemble du langage LDS est réaliste et suffisamment expressif. SCEC a pu détecter une erreur très subtile du compilateur CAdvanced. Ceci prouve d'une part, l'efficacité de notre approche et d'autre part, que le développement de ce type de vérificateur en vaut la peine. Nous sommes persuadés que notre concept de preuve d'équivalence peut se généraliser aux langages dédiés, dont les compilateurs sont nécessairement moins robustes, que les compilateurs classiques.