Experiments on automation of formal verification of devices at the binary level

THOMAS LACROIX

Master in Computer Science
Date: 24 mai 2019
Supervisor: Mads Dam
Examiner: TODO
Computer Science department - INSA Lyon
Host company: Department of Theoretical Computer Science - KTH

Résumé

With the advent of virtualization, more and more work is put into the verification of hypervisors. Being low level softwares, such verification should preferably be performed at binary level. Binary analysis platforms are being developed to help perform these proofs, but a lot of the work has to be carried out manually.

In this thesis, we focus on the formal verification of a Network Interface Controller (NIC), more specifically we look at how to automate and reduce the boilerplate work from an existing proof. We base our work on the HolBA platform, its hardware-independent intermediate representation language BIR and supporting tools, and we experiment on how to perform this proof by leveraging existing tools.

We first replaced the existing NIC model written in HOL4 to an equivalent one written using BIR, enabling the use of HolBA tools. Secondly, we developed some visualization tools to help navigate and gain some insight in the existing proof and its structure. Thirdly, we experimented with the use of Hoare triples in conjunction with an SMT solver to perform contract verification. Finally, we proved a simple contract written in terms of the formal NIC model on the BIR implementation of this model, unlocking the way of performing more complex proofs using the HolBA platform.

Keywords : binary analysis, formal verification, proof producing analysis, theorem proving

Résumé

Avec la démocratisation de la virtualisation, de plus en plus d'efforts sont consacrés à la vérification des hyperviseurs. S'agissant de logiciels de bas niveau, une telle vérification devrait de préférence être effectuée au niveau binaire. Des plates-formes d'analyse binaire sont en cours de développement pour aider à réaliser ces preuves, mais une grande partie du travail doit encore être effectuée manuellement.

Dans cette thèse, nous nous concentrons sur la vérification formelle d'un Contrôleur d'Interface Réseau (NIC), plus spécifiquement sur la manière d'automatiser et de réduire le travial d'une preuve existante. Nous nous basons sur la plate-forme HolBA, son langage de représentation intermédiaire indépendant du matériel, BIR et ses outils de support, et nous nous intéressons à la manière de réaliser cette preuve en utilisant des outils existants.

Nous avons d'abord remplacé le modèle NIC existant écrit en HOL4 par un modèle équivalent écrit en BIR, permettant ainsi l'utilisation des outils de HolBA. Deuxièmement, nous avons développé des outils de visualisation pour vous aider à naviguer et à mieux comprendre la preuve existante et sa structure. Troisièmement, nous avons expérimenté l'utilisation des triplets de Hoare en conjonction avec un solveur SMT pour effectuer une vérification par contrat. Enfin, nous avons prouvé un contrat simple écrit en termes du modèle formel du NIC sur l'implémentation de ce modèle en BIR, ouvrant la voie à la réalisation de preuves plus complexes avec la plate-forme HolBA.

Mot-clés : binary analysis, formal verification, proof producing analysis, theorem proving

Todo list

Include :a) Git workflow for a team ;b) LogLib (and tracing in general) ;c) CI to track regressions + static analysis ;

Table des matières

Chapitre 1

Introduction

This chapter serves as an introduction to the degree project and presents the background of the work along with this thesis objective. Delimitations to the project and the choice of methodology are also discussed.

1.1 Background

1.2 Intended readers

1.3 Thesis objective

1.4 Delimitations

1.5 Choice of methodology

Chapitre 2

Definitions and relevant theory

This chapter intends to lay the concepts and theory that are essential to the reader in order to understand the problem that this degree project aims to explore. This includes an overview of virtualization and hypervisors, a presentation of Interactive Theorem Proving and formal proofs, relevant theory about Hoare Triple and Weakest Precondition analysis, and an introduction of the ***HOL4 Binary Analysis Platform (HolBA)*** *framework.*

Chapitre 3

NIC model

Chapitre 4

Proving properties

4.1 Contract based verification

4.1.1 Hoare triples

Contract based verification is a powerful approach for verifying programs. For a given program $prog$ consisting of a list of instructions and two predicates P and Q called respectively pre- and postcondition, a Hoare triple $\{P\}\ prog\ \{Q\}$ states that when executing the program $prog$ from a state S terminates in a state S', if P holds in S then Q will hold in S' (Equation 4.1). Hereafter, we assume programs and states to be well-typed.

$$\{P\}\ prog\ \{Q\} \triangleq S' = exec(S, prog) \implies P(S) \implies Q(S') \tag{4.1}$$

For example, $\{P\}\ \varnothing\ \{P\}$ holds because an empty program doesn't change the state of the execution. $\{n = 1\}\ n := n + 1\ \{even(n)\}$, with $n \in \mathbb{N}$, holds because $1 + 1 = 2$, which is even.

In order to perform the verification, the Hoare logic introduces a set of axioms describing the effect of each instruction of a given language over the execution state [1]. For an assignment $x := f$ where x is a variable identifier and f an expression without side-effects, Equation 4.2 defines the axiom of assignment, where $P[f/x]$ denotes the subtitution of all occurences of x by f in P.

$$\{P[f/x]\}\ x := f\ \{P\} \tag{4.2}$$

4.1.2 Weakest precondition derivation

While Hoare logic introduces sufficient preconditions, Dijkstra introduced the concept of nessecary and sufficient preconditions, called "weakest" preconditions. Such weakest preconditions can be automatically derived from a program $prog$ and a postcondition Q. Let's call $WP(prog, Q)$ such a weakest precondition. Then, from Equation 4.1 follows :

check that equation 4.3 is valid

$$\forall(prog, Q), \{\, WP(prog, Q)\}\ prog\ \{Q\} \tag{4.3}$$

For the program $n := n + 1$ mentioned above, we can generate the weakest precondition for the postcondition $even(n)$. First, we can rewrite $even(n)$ as $n\ MOD\ 2 = 0$ with MOD denoting the arithmetic modulo. Then, we derive the weakest precondition of the statement $n := n + 1$ by transforming the predicate $n\ MOD\ 2 = 0$ by substituting all occurences of n by $n + 1$:

$$WP(\text{“}n := n + 1'', n\ MOD\ 2 = 0) = (n + 1\ MOD\ 2 = 0) \tag{4.4}$$

From the properties of the modulo, we can simplify $n + 1\ MOD\ 2 = 0$ to $n\ MOD\ 2 = 1$ or $odd(n)$. Therefore, $\{odd(n)\}\ n := n + 1\ \{even(n)\}$, i.e. incrementing the value of an odd integer variable by one makes it even.

While the triple $\{n = 1\}\ n := n + 1\ \{even(n)\}$ uses a sufficient precondition for establishing its postcondition, the triple $\{odd(n)\}\ n := n + 1\ \{even(n)\}$ uses the weakest precondition. The later being the weakest precondition of the former, the two contracts are in relation :

$$n = 1 \implies odd(n) \tag{4.5}$$

More generally, for a triple $\{P\}\ prog\ \{Q\}$ to hold, P must be stronger than the weakest precondition, i.e. we need to prove that $P \implies WP(prog, Q)$.

Here and above, should we mention/explain termination ?

$$(P \implies WP(prog, Q)) \implies \{P\}\ prog\ \{Q\} \tag{4.6}$$

4.1.3 Using SMT solvers to prove contracts

From Equation 4.6 we see that, in order to prove that a triple $\{P\}\ prog\ \{Q\}$, we need to prove $P \implies WP(prog, Q)$. While multiple methods exist to perform such proofs, **SMT** solvers offer a convient and automatic solution.

Satisfiability Modulo Theories (**SMT**) problem is a decision problem for logical formulas with respect to combinations of background theories such as arithmetic, bit-vectors, arrays, and uninterpreted functions [2]. **Satisfiability Modulo Theories (SMT)** problem is a generalisation of **Boolean SATisfiability Problem (SAT)** problem supporting more theories. When given a formula, a **SMT** solver decides if the formula is satisfiable, i.e. if there exist a valuation of its variables where the formula evaluates to true. As a **SMT** solver can fail to decide a given instance, there are three possible outputs : "satisfiable", "unsatisfiable" and "unknown". Another useful feature of some **SMT** solvers is the ability to ask for a satisfying model, which represents a counter-example of a false predicate.

A predicate P holds if it evaluates to true for all possible values of its variables. Alternatively, the negation of a predicate $\neg P$ holds if there exist no valuation of its variables where the predicate evaluates to true, i.e. if the instance is unsatisfiable. Therefore, if a **SMT** solver report that $\neg P$ is "unsatisfiable", then P holds.

isn't that the definition of tautologies ?

Using De Morgan's Laws, we know that $\neg(P \implies WP) \equiv (P \wedge \neg WP)$. Therefore, proving that $\neg(P \implies WP)$ is "unsatisfiable" using an **SMT** solver can be seen as proving that there exist no model where P holds and WP doesn't.

4.1.4 Contract based verification in HolBA

HolBA provides a tool for automatically deriving weakest preconditions on loop-free **BIR** programs whose control flow can be statically identified [3].

4.2 Using SMT solvers to prove contracts

4.3 Implementation of a non proof-producing WP library

4.3.1 Pretty-printing to visualize huge BIR expressions

4.3.2 Testing the library

4.4 Simple automatized proofs on the NIC model

4.5 Trustful analysis on the NIC model

Chapitre 5

Conclusions

5.1 Results

5.2 Discussion

5.3 Future work

Bibliographie

[1] C. A. R. Hoare. « An Axiomatic Basis for Computer Programming ». In : *Commun. ACM* 12.10 (oct. 1969), p. 576–580. ISSN : 0001-0782. DOI : 10.1145/363235.363259. URL : http://doi.acm.org/10.1145/363235.363259 (visité le 24/05/2019).

[2] Nikolaj Bjørner et al. *Programming Z3*. Microsoft Research. Avr. 2019. URL : https://theory.stanford.edu/~nikolaj/programmingz3.html (visité le 24/05/2019).

[3] Andreas Lindner, Roberto Guanciale et Roberto Metere. « TrABin : Trustworthy analyses of binaries ». In : *Science of Computer Programming* 174 (avr. 2019), p. 72–89. ISSN : 0167-6423. DOI : 10.1016/j.scico.2019.01.001. URL : http://www.sciencedirect.com/science/article/pii/S0167642318301667 (visité le 05/02/2019).

Annexe A

Something Extra