Model checking and test generation: a combined approach to software verification

Software systems are driving our life. Even critical infrastructures and safety-critical systems are becoming more and more software driven, where ensuring correctness is an important task. Various approaches exist to find software bugs: formal verification examines the mathematical model of the software and proves the absence of bugs. Formal verification is a computationally expensive task as it explores the possible states of the software, but the state space of even simple software systems can be huge or even infinite. Over the past decade numerous approaches were designed, that were able to lessen the algorithmic complexity of such methods, and the technological development helped to reduce the required computation time even further, however such software still exists, that still cannot be formally verified. On the other hand, software testing is an efficient technique to find bugs It is computationally cheaper than formal verification, and it is widely used by the industry. However testing alone cannot prove correctness.

The goal of my work is to combine formal verification with testing to exploit the advantages of both approaches. I introduce a new algorithm that uses various abstraction-based model checking techniques from the literature to explore the behavior of the software. If the formal verification algorithm can prove the correctness of the system, then the system is verified. However, as it usually happens to real-life problems, formal verification rarely can succeed. In such case, testing needs to be utilized. My algorithm stops the verification after a certain threshold is reached (time or memory limit), and applies test generation. My novel algorithm exploits the result of the formal verification and uses the information gathered during state space traversal: the test generation focuses only on those part of the state space, which was not fully explored during the model checking. I also introduce improvements to find programming errors, that can not be found by formal verification.

I will investigate my algorithm in benchmark models from the literature. I hope that the novel combination of formal verification and testing will improve the quality of safety-critical software systems.

Modellellenőrzés és tesztelés: egy kombinált megközelítés szoftverek verifilálására

Különböző szoftveres rendszerek életünk szerves részét képezik. Mára még a biztonság-kritikus rendszerekben is egyre inkább jelen vannak különböző, szoftvert használó komponensek, ahol a helyesség biztosítása jelentős feladat. Különböző megközelítések léteznek szoftverekben való hibakeresésre. Egyrészről a formális verifikáció, ami a szoftver matematikai modelljét vizsgálja, és matematikailag bizonyítja a hibák nemlétét. A formális verifikáció egy számításigényes feladat, hiszen megvizsgálja a szoftver összes lehetséges állapotát, viszont még a legegyszerűbb programoknak is hatalmas lehet az állapottere, ha egyenesen nem végtelen. Az elmúlt évtizedben számos megközelítés született, amik csökkenteni tudták ezen feladatok komplexitását, illetve a technológiai fejlődésnek köszönhetően a számításokhoz szükséges idő is csökkent. Viszont mindezek ellenére vannak olyan szoftverek, amiknek hibamentes működését még mindig nem tudjuk formális verifikációval igazolni. Egy másik merőben más megközelítés a tesztelés, ami hatékonyan képes hibákat találni a meglévő rendszerekben. Számításigényét tekintve sokkal szerényebb a formális verifikációnál, és az iparban elterjedten használt, viszont önmagában a helyes működés igazolására nem használható.

A munkám célja, hogy ezt a két különböző megközelítést kombináljam, ötvözve a két módszer előnyeit. Bemutatok egy új algoritmust, ami több különböző absztrakció alapú modellellenőrzési technikát használ a szoftver viselkedésének vizsgálatára. Ha ezek a módszerek képesek bizonyítani a szoftver helyességét, a verifikáció sikeres. Ellenben, ha a valós életben sokkal gyakrabban előforduló eset áll fent, név szerint, hogy a formális módszerek sem hiba létét, sem annak hiányát nem képesek igazolni, tesztelési technikák lesznek alkalmazva. Ehhez az algoritmus leállítja a formális verifikációt egy bizonyos határ után (ami lehet idő vagy memória korlát), és teszteket generál. A tesztek generálásához az algoritmus felhasználja az állapottér bejárása során gyűjtött információkat, és az állapottér csak azon részét teszteli, melynek helyességét a formális verifikáció nem tudta igazolni. Ezen kívül külön fókuszálok olyan hibák megtalálására, amelyeket a formális verifikáció önmagában nem tud megtalálni.

Az algoritmusomat az irodalomban fellelhető referencia modellekhez fogom mérni. Remélem, hogy ez az újszerű kombinációja a formális verifikációnak és tesztelésnek segíteni fog a biztonság-kritikus szoftverek minőségének javításában.