Kivonat

Egyre nagyobb szerepet kapnak az életünkben a különböző szoftvereszközök. Ma már olyan, biztonságkritikus alkalmazási területeken is elterjedten alkalmaznak szoftvereket, ahol egy esetleges meghibásodás végzetes következményekkel is járhat: szoftverhibák hatására vagyonok úszhatnak el, repülőgépek zuhanhatnak le, atomerőművek állhatnak le. Az ilyen, kritikus szoftverek helyességének biztosítása tehát kiemelten fontos feladat.

A szoftverek megfelelő működésének ellenőrzése a verifikációjuk során történik. Kritikus szoftverek ellenőrzésére gyakran alkalmaznak formális módszereket, melyekkel – ellentétben a hagyományos ellnőrzési módszerekkel (pl. tesztelés) – nemcsak hibák jelenléte mutatható ki, hanem akár a rendszer helyessége is bizonyítható. Az egyik legelterjedtebben alkalmazott formális módszer a modellellenőrzés, ahol a szoftver állapotterének bejárásával próbáljuk a vele szemben megfogalmazott formális követelmények teljesülését bizonyítani, vagy éppen cáfolni.

Az IC3 egy korszerű, eredetileg hardvermodellek ellenőrzésére alkalmas modellellenőrző algoritmus. Az IC3 algoritmus egyedi jellemzője, hogy a rendszer helyességének ellenőrzését inkrementálisan, számos egyszerű lemma indukciós bizonyításán keresztül végzi. Amennyiben a rendszer helyes, a felfedezett lemmák összessége a rendszer helyességének bizonyítékát (az állapottér egy biztonságos felülbecslését) adják; ellenkező esetben pedig a keresést egy ellenpélda irányába irányítják, ezáltal hatékonyan vezérelve az állapottérfelderítést. Az eredeti algoritmus hátránya, hogy elsősorban véges állapotterű rendszerek (pl. sorrendi hálózatok) hatékony kezelését teszi lehetővé. Az algoritmus így jelenleg is aktív kutatás tárgyát képezi, és számos kiterjesztéssel bír.

Dolgozatomban az IC3 algoritmusnak egy olyan kiterjesztését vizsgálom, amely a végtelen állapottér hatékony kezeléséhez (implicit) predikátumabsztrakciót használ. Ez a módszer a végtelen állapotteret véges sok partícióra osztja fel a változókon értelmezett logikai állítások (predikátumok) mentén. A partíciók egy véges absztrakt állapotteret hoznak létre, melynek állapotait az határozza meg, hogy mely predikátumok teljesülnek. A futás során az algoritmus új predikátumokat tanul, ezzel szükség szerint pontosítja a képét a rendszerről, mégis megtartja az absztrakció általánosságát.

Annak érdekében, hogy az algoritmus alkalmas legyen programokat leíró vezérlésifolyam-automaták ellenőrzésére, egy olyan transzformációt is megadok, amely az automata alapján egy vele ekvivalens viselkedésű szimbolikus tranzíciós rendszert állít elő. Az algoritmust ezen a szimbolikus tranzíciós rendszeren futtatva lehetőség nyílik az automata helyességének ellenőrzésére.

Az algoritmus, illetve a transzformáció implementációját a Theta nyílt forráskódú modellellenőrző keretrendszer moduljaként készítem el, ami további kiterjesztési lehetőségeket biztosít. Az implementált algoritmus különböző konfigurációinak hatékonyságát mérésekkel vizsgálom.

Abstract

Software plays an ever increasing role in our lives. Nowadays, software is used even in safety critical fields where a potential failure can lead to significant consequences: software failures can cause fortunes to go to waste, airplanes to crash, or nuclear power plants to stop. Thus ensuring the correctness of such critical software is a crucial task.

Verification is the process of checking that the software operates as expected. Formal methods are often used to verify critical systems, which – as opposed to traditional verification methods (such as testing) – is not only able to show the presence of errors, but also to prove their absence. One of the most widely applied formal method is model checking, which constitutes traversing the state space of the software to prove or disprove formal requirements against the system.

IC3 is a state-of-the-art model checking algorithm, originally developed for checking hardware models. The unique property of IC3 is that it proves the correctness of the system in incremental steps by proving several simple lemmas about the system. If the system is correct, the proven lemmas collectively form a proof of that fact (they specify a safe overapproximation of the state space). If the system is not correct, the lemmas direct the search towards a counterexample, thereby making the traversal more efficient. The original algorithm is designed to check finite state systems, and is not effective for infinite state systems. Therefore the algorithm is still the subject of active research and has several extensions.

In my thesis I examine an extension to IC3 that uses (implicit) predicate abstraction to handle infinite state spaces efficiently. This method divides the state space into a finite number of partitions with respect to a set of predicates over the system variables. The partitions form a finite abstract state space, where each state is defined by which of the predicates hold and which do not. During its run, the algorithm learns new predicates to refine its abstract image of the system, while still maintaining the coarseness of the abstraction.

In order to check control flow automata which describe programs, I present a transformation that creates a symbolic transition system for a control flow automaton that behaves equivalently to the automaton. The correctness of the automaton can be checked by running the algorithm on this symbolic transition system.

I implement the algorithm and the transformation as part of the open source model checking framework Theta, which allows for additional ways of extending it. Finally, I evaluate the efficiency of the various configurations of the implemented algorithm.