Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Verification of Timed Automata by CEGAR-Based Algorithms

Scientific Students' Associations Report

Author:

Rebeka Farkas

Supervisors:

András Vörös
Tamás Tóth
Ákos Hajdu

2016.

Contents

Kivonat A napjainkban egyre inkább elterjedő biztonságkritikus rendszerek hibás működése súlyos károkat okozhat, emiatt kiemelkedően fontos a matematikailag precíz ellenőrzési módszerek alkalmazása a fejlesztési folyamat során. Ennek egyik eszköze a formális verifikáció, amely már a fejlesztés korai fázisaiban képes felfedezni tervezési hibákat. A biztonságkritikus rendszerek komplexitása azonban gyakran megakadályozza a sikeres ellenőrzést, ami különösen igaz az időzített rendszerekre : akár kisméretű időzített rendszereknek is hatalmas vagy akár végtelen állapottere lehet. Ezért különösen fontos a megfelelő modellezőeszköz valamint hatékony verifikációs algoritmusok kiválasztása. Az egyik legelterjedtebb formalizmus időzített rendszerek leírására az időzített automata, ami a véges automata formalizmust óraváltozókkal egészíti ki, lehetővé téve az idő múlásának reprezentálását a modellben.

Formális verifikáció során fontos kérdés az állapotelérhetőség, amely során azt vizsgáljuk, hogy egy adott hibaállapot része-e az elérhető állapottérnek. A probléma komplexitása már egyszerű (diszkrét változó nélküli) időzített automaták esetén is exponenciális, így nagyméretű modellekre ritkán megoldható. Ezen probléma leküzdésére nyújt megoldást az absztrakció módszere, amely a releváns információra koncentrálva próbál meg egyszerűsíteni a megoldandó problémán. Az absztrakció-alapú technikák esetén azonban a fő probléma a megfelelő pontosság megtalálása. Az ellenpélda vezérelt absztrakciófinomítás (counterexample-guided abstraction refinement, CEGAR) iteratív módszer, amely a rendszer komplexitásának csökkentése érdekében egy durva absztrakcióból indul ki és ezt finomítja a kellő pontosság eléréséig.

Munkám célja hatékony algoritmusok fejlesztése időzített rendszerek verifikációjára. Munkám során az időzített automatákra alkalmazott CEGAR-alapú elérhetőségi algoritmusokat vizsgálom és közös keretrendszerbe foglalom, ahol az algoritmusok komponensei egymással kombinálva új, hatékony ellenőrzési módszerekké állnak össze. Az irodalomból ismert algoritmusokat továbbfejlesztettem és hatékonyságukat mérésekkel igazoltam.

Abstract Nowadays safety-critical systems are becoming increasingly popular, however, faults in their behavior can lead to serious damage. Because of this, it is extremely important using mathematically precise verification methods during their development. One of these methods is formal verification that is able to find design problems since early phases of the development. However, the complexity of safety-critical systems often prevents successful verification. This is particularly true for real-time systems: even small timed systems can have large or even infinite states pace. Because of this, selecting an appropriate modeling formalism and efficient verification algorithms is very important. One of the most common formalism for describing timed systems is the timed automaton that extends the finite automaton with clock variables to represent the elapse of time.

When applying formal verification, reachability becomes an important aspect – that is, examining whether or not the system can reach a given erroneous state. The complexity of the problem is exponential even for simple timed automata (without discrete variables), thus it can rarely be solved in case of large models. Abstraction can provide assistance by attempting to simplify the problem to solve while focusing on the relevant information. In case of abstraction-based techniques the main difficulty is finding the appropriate precision. Counterexample-guided abstraction refinement (CEGAR) is an iterative method starting from a coarse abstraction and refining it until the sufficient precision is reached.

The goal of my work is to develop efficient algorithms for verification of timed automata. In my work I examine CEGAR-based reachability algorithms applied to timed automata and I integrate them to a common framework where components of different algorithms are combined to form new and efficient verification methods. I improved known algorithms and proved their effectivity by measurements.

TODO: Ákos-javítások

Chapter 1

Introduction

TODO: Abstract+ kis módosítás

Chapter 2

Background

TODO: Ákos dipterv

2.1 Mathematical logic

2.1.1 Zeroth order logic

SAT

2.1.2 First order logic

SMT

2.2 Formal verification

TODO: Importance, etc.

2.2.1 Timed automata

TODO: Modeling formalisms, timed systems, etc.

Basic Definitions

In order to properly define timed automata, first the idea of *clock variables* must be explained. In case of systems with discrete variables, the values of the variables always remain the same betrween two modifications. However, this is not the case for clock variables (clocks, for short). Even when a system stays in one state, the value of clocks are continuously and steadily increasing. Naturally, their values can be modified, but the only allowed operation on clock variables is *reset*. Reseting a clock means assigning its value to a specific integer (often, that integer can only be 0). It's an instantaneous operation, after which the value of the clock will continue to increase.

Hereinafter follows some basic definitions that are closely related to clock variables and timed automata.

A *valuation* $v(\mathcal{C})$ assigns a non-negative real value to each clock variable $c \in \mathcal{C}$, where $\mathcal{C}$ denotes the set of clock variables.

In other words a valuation defines the values of the clocks at a given moment of time. The term *valuation* can also be used for discrete variables.

A *clock constraint* is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ (*difference constraint*), where $x, y \in \mathcal{C}$ are clock variables, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. $\mathcal{B}(\mathcal{C})$ represents the set of clock constraints.

In other words a clock constraint defines upper and lower bounds on the values of clocks (or differences of clocks, in case of difference constraints). Bounds are always integer numbers. Clock constraints are used in guards and invariants of timed automata to control the behaviour by only allowing certain operations if the current valuation satisfies the consraints.

A *timed automaton* extends a finite automaton with clock variables. It can be defined as follows.

A *timed automaton* $\mathcal{A}$ is a tuple $\langle L, l_0, E, I \rangle$ where

- L is the set of locations,

- $l_0 \in L$ is the initial location,

- $E \subseteq L \times \mathcal{B}(\mathcal{C}) \times 2^{\mathcal{C}} \times L$ is the set of edges and

- $I : L \to \mathcal{B}(\mathcal{C})$ assigns invariants to locations. Invariants can be used to ensure the progress of time in the model. [**bengtsson2004timed**]

Graphically a timed automaton can be represented as a labeled graph where the vertices are the locations labelled with their corresponding invariants, and the edges are the automaton's edges, that are defined by the source location, the guard (represented by a clock constraint), the set of clocks to reset, and the target location.

TODO: példa

A state of $\mathcal{A}$ is a pair $\langle l, v \rangle$ where $l \in L$ is a location and v is the current valuation satisfying $I(l)$. In the initial state $\langle l_0, v_0 \rangle$ v_0 assigns 0 to each clock variable.

Two kinds of operations are defined. The state $\langle l, v \rangle$ has a *discrete transition* to $\langle l', v' \rangle$ if there is an edge $e(l, g, r, l') \in E$ in the automaton such that

- v satisfies g,

- v' assigns 0 to any $c \in r$ and assigns $v(c)$ to any $c \notin r$, and

- v' satisfies $I(l')$.

The state $\langle l, v\rangle$ has a *time transition* (or delay, for short) to $\langle l, v'\rangle$ if

- v' assigns $v(c) + d$ for some non-negative d to each $c \in \mathcal{C}$ and
- v' satisfies $I(l)$.

There are many variations of timed automata (e.g. this definition only allows to reset clocks to 0, however, resets to greater integers will appear later in this paper). Most of them such as network automata, synchronization, and urgent locations can be easily transformed into conventional timed automata, but this is not always the case. The idea to allow discrete variables as well as clock variables arises simply. Bool, integer, rational, or even self-described typed variables prove useful, but may result in a formalism with bigger expressive power that that of the conventional timed automaton. This becomes important when one wants to analyze a system.

2.2.2 Reachability

TODO: Importance, basic algorithms (statespace exploration, SAT based solution, bounded stuff + examples), TA reachability + examples

Timed automaton reachability

2.2.3 CEGAR

Abstraction

Idea, usefulness, Times automata - zones, variables, activity, etc.

CEGAR-loop

Idea, Cegar-loop, basic cegar ideas (variable-based, statespace refinement, etc.)

Chapter 3

Configurable Timed CEGAR

This chapter presents a configurable framework for CEGAR-based reachability analysis of timed automata.

3.1 Generic CEGAR Framework

The key idea of the framework is to provide various implementations of each phases of the CEGAR-loop, by using correspondent parts of CEGAR-based reachability algorithms. Most of these algorithms already exist for other formalisms, and they have to be adapted to timed automata, but some of them are new approaches. The implemented modules can then be combined (that is, the implementation of each phases can be provided by different algorithms) to form new algorithms, and chosing the most effective parts of the original algorithms can result in an even more effective algorithm then the original ones.

The architecture of the framework is illustrated on **TODO:** ábra: (két résszel az automatáshoz és az állapottereshez) amin látszanak hogy pontosan mik lesznek a dobozok (milyen interfészek) és mi megy köztük a nyilakon, stb. As one can see, there are two different realization of the CEGAR-loop. The reason for this is that not all implementations of the CEGAR-phases can be used interchangably, since there are two distict ways CEGAR-loop can be applied to timed automata. The key difference is the basis of the refinement. While the first approach **TODO:** ábra a) részét referálni starts from a pure automaton (without any clock variables) and extends the current automaton with some clocks in each iteration **TODO:** háttérismereteknél referálni - and thus refines the *automaton*, the other **TODO:** ábra b) részét referálni is based on the refinement of the *statespace* itself. Because of this, only algorithms of the same approach can be combined.

3.1.1 Automaton-based refinement

TODO: Fig ... depicts the architecture... The initial abstraction is a finite automaton that is derived from the original timed automaton by removing all clock variables and clock constraints.

In each iteration of the CEGAR-loop, the task of the model checking phase is to determine whether the error location is reachable in the current automaton and provide a trace (counterexample) if there is one. Therefore, the implementation should be a reachability-checking algorithm for timed automata that can find a trace to the location.

The task of the analysis phase is to check if the found trace is feasible in the original automaton and if it isn't, provide a set of clock variables that can then be added to the automaton (with the clock constraints they appear in) so that the model checker won't find this counterExample again. his is quite a complex task and therefore there aren't many implementations of it.

Finally, the only task of the refinement phase is to refine the current abstraction of the automaton, by extending it with the given set of clock variables (and the constraints they appear in). The task is straightforward, and so this part of the CEGAR-loop has only one implementation.

TODO: A pseudocode is provided to demonstrate implementability.

3.1.2 Statespace-based refinement

TODO: Fig ... depicts the architecture... In this case the initial abstraction is not as obvious. It has to be some kind of the

TODO: CEGAR dobozok, interfészek, cserélgethetőség, stb. ábrával + egy mini pszeudokód

3.2 Modules

Implementált modulok felsorolva, utalással az előző fejezet algoritmusaira. + példák, pszeudokód

3.3 Combined Algorithms

A fenti modulok kombinálhatósága. Ekészült algoritmusok.

Chapter 4

Implementation

4.1 Environment

TODO: TTMC bemutatása,stb

4.2 Measurements

4.2.1 Objectives

TODO: Célok ismertetése, mérések bemutatása. Mit akarunk mérni, mivel fogjuk összehasonlítani, milyen bemeneteken, és miért.

4.2.2 Inputs

TODO: Uppaal inputok, stb.

4.2.3 Results

TODO: Grafikonok + mit mértünk épp, mivel, mi lett az eredménye

4.2.4 Evaluation

TODO: Miérések eredményének összesítése, mit tudtunk meg ebből.

Chapter 5

Related Work

TODO: Milyen más Timed CEGAR megközelítések vann, és ehhez képest a miénk miben más, és főleg mibven jobb.

Chapter 6

Conclusions

TODO: Ha van valami nagyobb/meglepőbb eredmény, azt lehet hangsúlyozni.

6.1 Contribution

TODO: Szokásos pontokba szedett, részletes kontribúcióismertetés.

6.2 Future work

TODO: predikátum interpolánssal + egyebek Pl. paraméteres, vagy Ákossal összedolgozós, stb.