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

Contents 3

Kivonat 5

Abstract 7

1 Introduction 1

2 Background 3
2.1 Timed Automata . . . 3
2.1.1 Basic Definitions . . . 3
2.1.2 Reachability Analysis . . . 4
2.2 CEGAR . . . 7

3 Configurable Timed CEGAR 9
3.1 Algorithms . . . 9
3.1.1 Adapted algorithms . . . 9
3.1.2 Solver-based approaches . . . 9
3.1.3 Statespace refinement . . . 9
3.1.4 Trace Activity . . . 9
3.2 Architecture . . . 9
3.2.1 Overview . . . 9
3.2.2 Modules . . . 9

4 Implementation 11
4.1 Environment . . . 11
4.2 Measurements . . . 11
4.2.1 Objectives . . . 11
4.2.2 Algorithms . . . 11
4.2.3 Results . . . 11
4.2.4 Evaluation . . . 11

5 Related Work **13**

6 Conclusions **15**
6.1 Contribution 15
6.2 Future work 15

References **17**

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: Ennek a szerkezete még nincs teljesen kitalálva

2.1 Timed Automata

2.1.1 Basic Definitions

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. For bravity, the term *clock* is used as a synonym of clock variable.

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.

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.

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* 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 $I(l)$.

2.1.2 Reachability Analysis

A *zone* is a set of nonnegative clock valuations satisfying a set of clock constraints. The set of all valuations reachable from a zone z by time transitions is denoted by $z^{\uparrow}$.

A *zone graph* is a finite graph consisting of $\langle l, z\rangle$ pairs as nodes, where $l \in L$ refers to some location of a timed automaton and z is a zone. Therefore, a node denotes a set of states. Edges between nodes denote transitions.

The construction of the graph starts with the initial node $\langle l_0, z_0\rangle$, where l_0 is the initial location and z_0 contains the valuations reachable in the initial location by time transitions. Next, for each outgoing edge e of the initial location (in the automaton) a new node $\langle l, z\rangle$ is created (in the zone graph) with an edge $\langle l_0, z_0\rangle \rightarrow \langle l, z\rangle$, where $\langle l, z\rangle$ contains the states to which the states in $\langle l_0, z_0\rangle$ have a discrete transition through e. Afterwards z is replaced by $z^{\uparrow}$. The procedure is repeated on every newly introduced node of the zone graph. If the states defined by a newly introduced node $\langle l, z\rangle$ are all contained in an already existing node $\langle l, z'\rangle$, $\langle l, z\rangle$ can be removed, and the incoming edge should be redirected to $\langle l, z'\rangle$.

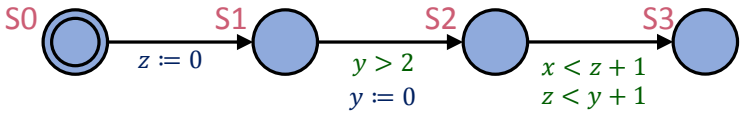

Figure 2.1 Timed automaton example

Example 2.1 For ease of understanding the algorithm is demonstrated on the automaton in Figure 2.1. The inital state is $\langle S0, z_0\rangle$ where z_0 is a zone containing only the initial valuation $v_0 \equiv 0$. The initial node is $\langle start, z_0^{\uparrow}\rangle$, where $z_0^{\uparrow}$ contains all states reachable form the initial state by delay. Since as time passes, the values of the three clocks will be incremented by the same value, x, y and z has the same value in each valuation contained by $z_0^{\uparrow}$. Since there is no invariant in location $S0$ the clocks can take any positive value. Because of this z_0 can be defined by the constraint $x = y = z$ (that is, $x - y = 0 \wedge y - z = 0$), and the initial node can be defined as $\langle S0; x = y = z\rangle$.

There is only one outgoing transition from the initial location and that resets z, resulting in the zone defined by $x = y \wedge z = 0$, which transforms into $z \leq x = y$ when delay is applied. This means the next node of the graph can be defined as $\langle S1, z \leq x = y\rangle$. There is only one outgoing transition from the location $S1$ and it has guard $y > 2$. This means the transition is only enabled in the subzone $z \leq x = y > 2$ (that is $z \leq x \wedge x = y \wedge y > 2$). Th transition resets y resulting in the

zone $y = 0 \wedge z \leq x > 2$. Delay can be applied and the next node of the graph turns out to be $\langle S2, z \leq x \wedge y \leq z \wedge x - y > 2 \rangle$.

There outgoing transition from location $S2$ has a guard $x < z + 1 \wedge z < y + 1$ from which we can determine $x < y + 2$ contradicting the atomic constraint $x - y > 2$ in the reachable zone of location $S2$. Thus the transition is never enabled, and location $S3$ is unreachable.

The resulting zone graph is the following.

$$\langle S0; x = y = z \rangle \rightarrow \langle S1, z \leq x = y \rangle \rightarrow \langle S2, z \leq x \wedge y \leq z \wedge x - y > 2 \rangle$$

Unfortunately, it is possible that the graph described by the previous algorithm becomes infinite. Consider for example the automaton from [1] in Figure 2.2.

Example 2.2 Constructing the zone graph of this automaton starts similarly, with the node $\langle start, x = y \rangle$. After that both x and y are reset resulting in the zone defined by $x = y = 0$. Location $loop$ has an invariant $x \leq 10$ that limits the applicable delay to 10, resulting in $\langle loop, x = y \leq 10 \rangle$, where only the loop-transition is enabled.

Tthe transition resets x resulting in $\langle loop, x = 0 \wedge y = 10 \rangle$. Still only 10 units of delay is enabled, resulting in the node $\langle loop, x \leq 10 \wedge y - x = 10 \rangle$.

From this node, both transitions are enabled. The loop transition increases the difference between x and y yielding the new node $\langle loop, x \leq 10 \wedge y \leq 30 \wedge y - x = 20 \rangle$, while the other transition resets both clocks, resulting in the new node $\langle end, x = y \rangle$.

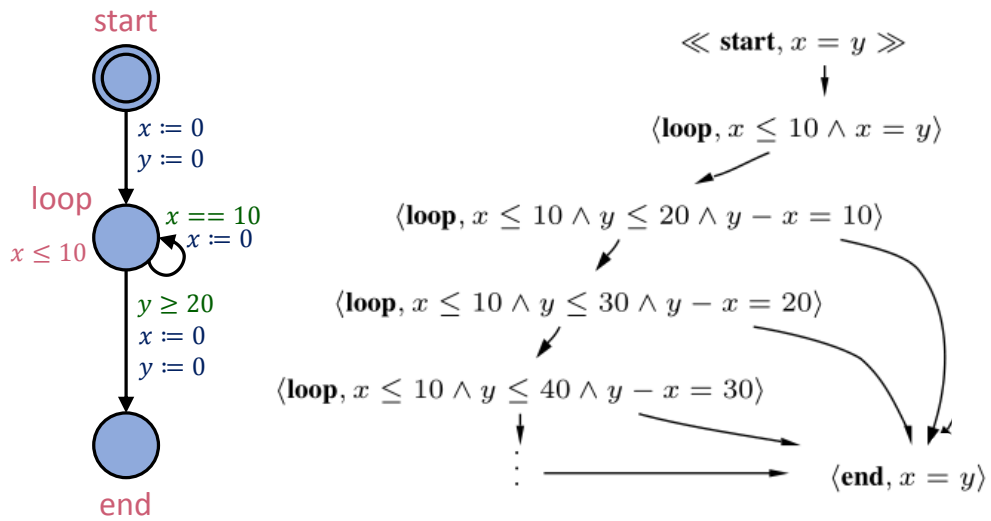

Figure 2.2 Timed automaton with infinite zone graph

As we take the new node containing the location $loop$, and apply the loop transition over and over, a new node is always constructed with the difference growing. On the other hand, the other transition always results in $\langle end, x = y \rangle$. Hence the (infinite) zone graph in Figure 2.2.

In order for the zone graph to be finite, a concept called *normalization* is introduced in [1]. Let $k(c)$ denote the greatest value to which clock c is compared in the automaton. For any valuation v such that $v(c) > k(c)$ for some c, each constraint in the form $c > n$ is satisfied, and each constraint in the form $c = n$ or $c < n$ is unsatisfied, thus the interval $(k(c), \infty)$ can be used as one abstract value for c.

Normalization is performed on $z^{\uparrow}$ (before inclusion is checked) in two steps. The first step is removing all constraints of the form $x < m, x \leq m, x - y < m, x - y \leq m$ where $m > k(x)$ (so that x doesn't have an upper bound), and the second step is replacing constraints of the form $x > m, x \geq m, x - y > m, x - y \geq m$ where $m > k(x)$ by $x > k(x), x \geq k(x), x - y > k(x), x - y \geq k(x)$ respectively (to define the new lower bounds).

Example 2.3 In the automaton depicted in Figure 2.2, $k(y) = 20$ (and $k(x) = 10$). This means the exact value of y doesn't really matter, as long as it is greater than 20 – the automaton will behave the exact same way if it is between 30 and 40, or if it is between 40 and 50. If we take this into consideration when constructing the zone graph, the zone $x \leq 10 \wedge y - x = 30$ can be normalized. In this zone, $y \geq 30 > k(y) = 20$, but $x \leq k(x)$. This means we only have to consider constraints bounding y. Implicitely $y \leq 40$ and $y - x \leq 30$. These constraints have to be removed from the zone. Similarly, $y \geq 30$ and $y - x \geq 30$ have to be replaced by $y \geq 20$ and $y - x \geq 20$. The resulting zone is $x \leq 10 \wedge y \geq 20 \wedge y - x \geq 20$. If we replace the original zone $x \leq 10 \wedge y - x = 30$ by this zone, and continue constructing the zone graph, the resulting graph is depicted in Figure 2.3.

Using normalization the zone graph is finite, but unreachable states may appear in it. If the automaton doesn't have any guard or invarint of the form $c_1 - c_2 < n$, the reachability of the location in question will be answered correctly. Otherwise, the algorithm may terminate with a false positive result.

Example 2.4 To demonstrate the incorrectness of the algorithm, consider again the automaton in Figure 2.1. Recall that the reachabble states of the automaton (by our calculations) were $\langle S0, x = y = z \rangle$, $\langle S1, z \leq x = y \rangle$ and $\langle S2, z \leq y \leq z \leq y \wedge x - y > 2 \rangle$ – $S3$ is unreachable. Applying normalization leaves the states $\langle S0, x = y = z \rangle$ and $\langle S1, z \leq x = y \rangle$ unchanged, but the normalizing the reachable state in $S2$ results in $\langle S2, z \leq y \leq z \leq y \wedge x - y > 1 \rangle$, where the guard can be satisfied, thus making $S3$ reachable.

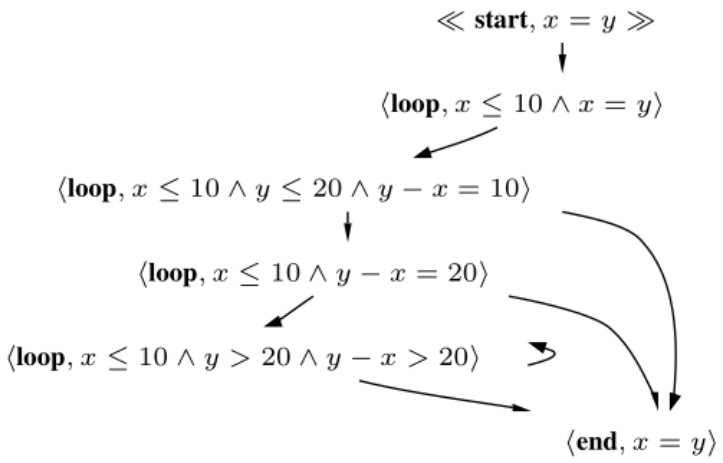

Figure 2.3 Finite zone graph

The operation *split* [1] is introduced to assure correctness. Instead of normalizing the complete zone, it is first split along the difference constraints, then each subzone is normalized, and finally the initially satisfied constraints are reapplied to each normalized subzone. The result is a set of zones (not just one zone like before), which means multiple new nodes have to be introduced to the zone graph (all with edges representing the same transition from the original node).

Example 2.5 To demonstrate the effects of split, let us construct the zone graph of the automaton of Figure 2.1. The original node remains $\langle S0, x = y = z\rangle$, but the next node is first split along the difference constraint $x - z < 1$. Instead of the node $\langle S1, z \leq x = y\rangle$, this time there are two nodes: $\langle S1, x = y \wedge x - z < 1\rangle$ and $\langle S1, x = y \wedge x - z \geq 1\rangle$.

From $\langle S1, x = y \wedge x - z < 1\rangle$, $\langle S2, x - z \leq 1 \wedge z - y \leq 1\rangle$ is reachable, where the transition to location $S3$ is not enabled because of the guard $x - z < 1$.

From $\langle S1, x = y \wedge x - z \geq 1\rangle$ the resulting zone after firing the transition is split along the constraint $z - y < 1$, resulting in nodes $\langle S2, x - z \geq 1 \wedge z - y < 1\rangle$, and $\langle S2, x - z \geq 1 \wedge z - y \geq 1\rangle$. The transition to $S3$ is not enabled in either nodes.

Applying split results in a zone graph, that is a correct and finite representation of the state space [1].

TODO: Műveletekről, komplexitásról beszélni!!!

2.2 CEGAR

The CEGAR approach introduced in [2] makes abstraction refinement a key part of model checking. The idea is illustrated on Figure 2.4.

First, an abstract system is constructed. The key idea behind abstraction is that the state space of the abstract system overapproximates that of the original one. Model checking is performed on this abstract model. If the target state is unreachable in the abstract model, it is unreachable in the original model as well. Otherwise the model checker produces a counterexample – a run where the system reaches the target state. In our case the counterexample is a sequence of transitions – i.e., a trace. Overapproximation brings such behaviors to the system that are not feasible in the original one. Because of this, the counterexample may not be a valid trace in the real system, so it has to be investigated. If it turns out to be a feasible counterexample, the target state is reachable. Otherwise the abstract system has to be refined. The goal of the refinement is to modify the abstract system so that it remains an abstraction of the original one, but the spurious counterexample is eliminated. Model checking is performed on the refined system, and the CEGAR loop starts over.

The algorithm terminates when no more counterexamples are found or when a feasible trace is given leading to the erroneous state.

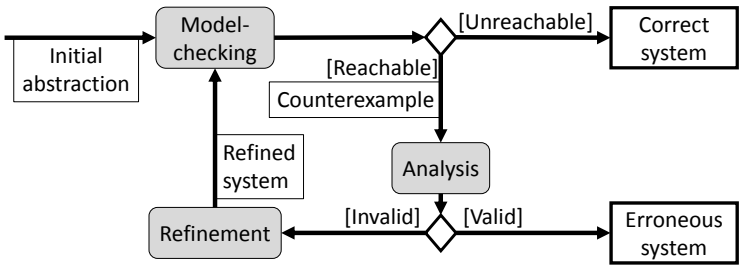

Figure 2.4 Counterexample-guided abstraction refinement

Chapter 3

Configurable Timed CEGAR

3.1 Algorithms

TODO: Algoritmusok, leírással, hol használják, hogyan lehet TA-ra alkalmazni, miért jó, stb. + hogyan fog beleilleszkedni a konfigurálható izébe (milyen dobozok)

3.1.1 Adapted algorithms

3.1.2 Solver-based approaches

3.1.3 Statespace refinement

3.1.4 Trace Activity

3.2 Architecture

3.2.1 Overview

TODO: CEGAR dobozok, interfészek, cserélgethetőség, stb. ábrával

3.2.2 Modules

TODO: Elkészült dobozok, intefészek, kombinálhatóság

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 Algorithms

TODO: Néhány kevert algoritmus bemutatása, érvelés, hogy miért tök jó, mit várunk a méréseknél.

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: Pl. paraméteres, vagy Ákossal összedolgozós, stb.

References

[1] Johan Bengtsson and Wang Yi. “Timed Automata: Semantics, Algorithms and Tools”. In: *Lectures on Concurrency and Petri Nets*. Vol. 3098. LNCS. Springer Berlin Heidelberg, 2004, pp. 87–124.

[2] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. “Counterexample-guided abstraction refinement for symbolic model checking”. In: *Journal of the ACM (JACM)* 50.5 (2003), pp. 752–794.