Towards Model Checking Capacity Timed Automata

Jonas Groth, Kasper Kastaniegaard, Søren Larsen, and Torkil Olsen

Aalborg University, Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg East, Denmark
{jgroth07,kkasta07,slars07,tolse09}@student.aau.dk
Supervisor: Mads Christian Olesen
René Rydhof Hansen

Abstract. We refine and study the class of *Capacity Timed Automata* (CTA) which is an extension of *Timed Automata* (TA) adding positive and negative cost rates on locations. The motivational example is the model of a robotic vacuum cleaner. During its run it must ensure that it does not reach a state in which it has insufficient power to return to the charging station. We use CTA to model the system and use the introduced capacity variables to represent the charge of the battery of the robotic vacuum cleaner. A proof is provided, showing that the reachability problem for CTA with one clock and two capacity variables is undecidable. The expressiveness of CTA is discussed in relation to related formalisms. Finally an over-approximating approach for verifying reachability properties of CTA is proposed.

1 Introduction

The field of energy or price modelling of real-time systems has been the focus of much research in recent years [BLR04],[BFL$^+$08],[BLM08]. This is an intriguing area to investigate as many of today's house hold devices contain some sort of electronics and software to make them act more intelligently. Many devices also operate with some sense of capacity e.g. battery, water reserve, storage capacity, etc.

This leads to the definition of a class of problems with the following characteristics: 1) There exists one or more resources that can increase or decrease in states as the system runs and time elapses. 2) The resources have a lower and upper bound e.g. a battery with zero and full charge. 3) The system is expected to run indefinitely while never exceeding the bounds on the resources.

To address this problem domain we refine the definition of the *Capacity Timed Automata* (CTA) formalism initially introduced in [LOHG10]. The formalism introduces the option of having multiple capacity variables with a defined lower and upper bound of the value for each variable. The capacity variables can be used to represent power level, water level, etc. Furthermore, the formalism allows specifying cost rates on locations which can be either positive or negative.

This is to express e.g. a system which charges a battery, represented as a capacity variable, in one location and consumes power from the battery in another location.

In this paper we discuss the expressiveness of CTA in relation to related formalisms for timed system which will be briefly described in Section 1.1. We will provide a proof showing that the reachability problem for CTA with one clock and two capacity variables is undecidable. Finally, we will discuss the data structure needed for an approximate model checking algorithm. We propose an initial over-approximating approach to verify reachability properties and conclude that the approach is not sufficient for all models. We refine the approach and define Capacity Timed Computational Tree Logic (CTCTL) which allows us to query CTA for properties. Furthermore, we prove that the behaviour of any time deterministic CTA is captured exactly by the over-approximation.

Throughout the paper we will be using a robotic vacuum cleaner as a motivating example to show the capabilities of the CTA formalism. The simple idea of a vacuum robot gives several interesting properties that can be modelled in CTA. The most obvious property of the robot is that it should be aware of it's battery so that it never runs so low on battery that it cannot make it back to the charging station and recharge the battery. The vacuum robot also has a small container in which it gathers the dust it picks up. This container will need to be emptied in some way and it would be best if the robot was aware of this and returned to its base when it is full to avoid a, perhaps hypothetical, situation where the robot would bust because the container had been overfilled.

1.1 Related Work

The model of *Timed Automata* (TA), introduced in [AD94], is a widely used formalism for modelling real-time systems. Several model checking tools exist for this formalism, including UPPAAL [UPP10] which relies on *Difference Bound Matrix* (DBM) for storing clock zones, allowing it to efficiently validate systems modelled as networks of TA.

Gerd Behrmann *et al.* [BLR04] introduce an extension of TA, namely *Priced Timed Automata* (PTA). The extension allows specifying a positive price on locations and edges. This introduces the possibility of doing *Cost Optimal Reachability Analysis* by measuring the accumulated cost of runs. This enables modelling areas such as power consumption in task scheduling systems. A special version of UPPAAL called *UPPAAL CORA* [COR10] has been developed specifically for this. Kim Larsen *et al.* [LR05] describe *Multi Priced Timed Automata* (MPTA), an extension of PTA, which extends the notion with multiple cost variables.

Weighted Timed Automata (WTA) is an extension of TA much like PTA, but allows both positive and negative weights on edges and locations. Patricia Bouyer *et al.* [BFL$^+$08] define WTA and several derivatives of WTA such as Weighted Timed Games which are proven undecidable under certain constraints.

Stopwatch Automata (SWA), as described in [CL00], is an extension of TA that allows for stopping and starting clocks which is equal to setting the rate of the clock to 0 to stop it and 1 to start it. The authors also prove that SWA

and *Linear Hybrid Automata* (LHA) are timed language equivalent and provide a method for translating LHA into SWA. SWA are unfortunately only semi-decidable and thus only approximate verification can be performed. The latest versions of UPPAAL are able to verify SWA.

Hybrid Automata (HA) can model systems with both discrete and continuous components as described in[CL00], [Hen96]. LHA [ACH$^+$95], mentioned above, is a restricted subclass of HA which only allows invariants, guards and activities to be linear expressions of the set of variables V. A *linear expression* is defined as $\phi(\bar{v})$ over V of the form $\Sigma a_i v_i$ with $a_i \in \mathbb{Z}$, $v_i \in V$.

The above described formalisms either do not provide the required expressiveness or are too general in their expressiveness. Therefore, we propose a new formalism that can bridge the gap between the simpler formalisms of TA, PTA and WTA and the more general and expressive formalisms such as SWA, LHA and HA. We want to provide a formalism that is powerful enough to address the characteristics of capacity problems while remaining restrictive enough to get verification results.

Outline of the paper. In Section 2 we define CTA along with parallel composition allowing networks of CTA. In Section 3 we prove undecidability of the reachability problem for CTA with one clock and two capacity variables. Section 4 describes the motivating example of a robotic vacuum cleaner in depth. Section 5 is a discussion of the expressiveness of CTA compared to the above mentioned formalisms. A discussion of a data structure and an approach to make approximate verification of CTA is described in Section 6 along with examples of how the approximation works. Finally, we conclude in Section 7.

2 Capacity Timed Automata

The CTA formalism is an extension of TA. The extension allows positive and negative costs in locations. This is based on the assumption that resources we want to model never change instantaneously but only as a result of time passing while remaining in some location. Before defining the CTA formalism, guards and invariants are defined as this is a preliminary for the definition of CTA.

Definition 1 (Guards and Invariants for Clocks and Capacity Variables). *A guard or invariant for the set of clocks X is defined by $\beta(X)$ which is the set of simple expressions over X of the form:*

$$g, g_1, g_2 ::= x \bowtie k \mid g_1 \wedge g_2$$

where $x \in X$, $\bowtie \in \{<, \leq, ==, \geq, >\}$ and $k \in \mathbb{N}_0$. A guard or invariant for the set of capacity variables C is defined by $\mathbf{\Delta}(C)$ which is the set of simple expressions over C of the form:

$$g, g_1, g_2 ::= c \bowtie k \mid g_1 \wedge g_2$$

where $c \in C$, $\bowtie \in \{<, \leq, ==, \geq, >\}$ and $k \in \mathbb{Z}$.

Definition 1 shows the definition of guards and invariants. Note that the definition for guards and invariants for clocks is the same as the definition given in [AILS08] for TA.

Definition 2 (Capacity Timed Automaton). *A Capacity Timed Automaton over a finite set of clocks X, a finite set of real-valued capacity variables C, a finite set of initial values O for each capacity variable, a function $B : C \longrightarrow \mathbb{Z} \times \mathbb{Z}$ that assigns bounds for each capacity variable s.t. $B(c_j) = [k_1, k_2]$, where $k_1, k_2 \in \mathbb{Z}$ and $c_j \in C$ and a finite set of actions $Act = N$, where N is the set of ordinary actions names, is a 5-tuple:*

$$A = (L, l_0, E, I, P)$$

where

- *L is the set of locations*
- *$l_0 \in L$ is the initial location*
- *$E \subseteq L \times \beta(X) \times \mathbf{\Delta}(C) \times Act \times 2^X \times L$ is the set of edges. An edge (l, gx, gc, a, r, l') is a transition consisting of a source location l, clock guards gx and capacity guards gc, an action a, a set $r \in 2^X$ of clocks that will be reset to 0 and a target location l'.*
- *$I : L \longrightarrow \beta(X) \times \mathbf{\Delta}(C)$ assigns clock and capacity invariants to locations*
- *$P : L \longrightarrow \mathbb{Z}^C$ assigns cost rates to locations s.t. $P(l)(c_j)$ is the cost rate of the capacity variable $c_j \in C$ in location l. Syntactically the rate is written as $\dot{c}_j$. When $\dot{c}_j$ is not defined on a location the cost rate 0 is assumed.*

It is worth noticing that the bounds given by B are strict bounds in the sense that a capacity variable may reach the defined bound but never exceed it. This is the definition most in tune with modern electronics. Most electronic devices that can recharge (notebooks, MP3 players, cell phones, etc.) have a built in circuit like the one described in [Pan07] to prevent overcharging. In an automaton this would translate to entering a new state in which no additional charge is added to the battery but the system is still relying on the power received from the charger.

Based on the above reasoning about bounds we can define bounds as implicit invariants on every location in the automaton. The invariants will be of the form $c_j \geq k_1$ and $c_j \leq k_2$ for any $c_j \in C$ and $B(c_j) = [k_1, k_2]$. These invariants make it impossible to stay in a location and perform a delay transition which would exceed the bounds defined for the capacity variable.

An edge in CTA between two locations contains guards for both clocks and capacity variables, $gx \in \beta(X)$ and $gc \in \mathbf{\Delta}(C)$ respectively. We may also specify a set of clocks, $r \in 2^X$, that will be reset to zero. Figure 1 illustrates an example of a CTA in its graphical representation and how guards and invariants on both capacity variables and clocks can be used.

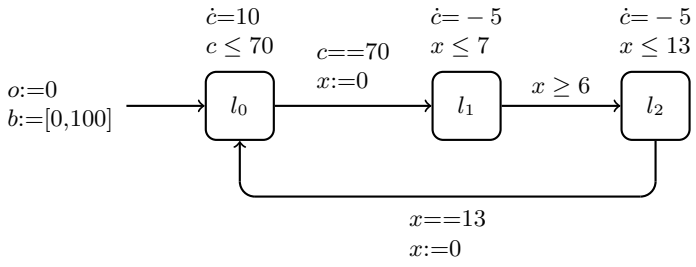


Figure 1. Example of a CTA.

To define the semantics of the CTA formalism we use a *Capacity Transition System* (CTS) shown in Definition 3. With CTS we add to the notion of the Labelled Transition System (LTS) [AILS08] formalism as done in [BLR04] for *Priced Transition System* (PTS).

Definition 3 (Capacity Transition System). *A Capacity Transition System is a tuple* $\mathcal{T} = (S, s_0, \Sigma, \longrightarrow)$*, where* S *is a (possibly infinite) set of states,* $s_0 \in S$ *is the initial state,* Σ *is a set of labels, and* $\longrightarrow$*:* $(S \times \Sigma \times S)$ *is the transition relation.*

With the definition of CTS we describe the semantics of a CTA in Definition 4. We denote a state in CTS by (l, v, q) where l is a location in CTA, v is a valuation of the clocks X and q is a valuation of the capacity variables C.

Definition 4 (Semantics of a Capacity Timed Automaton). *The semantics of a Capacity Timed Automaton* $A = (L, l_0, E, I, P)$ *over clocks* X*, capacity variables* C *and actions Act is given by a Capacity Transition System* $\mathcal{T} = (S, s_0, \Sigma, \longrightarrow)$*, where* $S = \{(l, v, q) \mid (l, v, q) \in L \times \mathbb{R}^X_{\geq 0} \times \mathbb{R}^C$ *and* $v, q \models I(l)$ *and* $q(c_j) \models B(c_j)$ *for all* $c_j \in C\}$ *is the set of states that satisfy the invariants,* $s_0 = (l_0, v_0, q_0)$ *is the initial state where* $\forall x \in X$ *we have* $x = 0 \in v_0$ *and* $q_0(c_j) = o_j$ *for all* $c_j \in C$ *and* $o_j \in O$*,* $\Sigma = Act \cup \mathbb{R}_{\geq 0}$ *is the set of labels, and* $\longrightarrow$ *consists of discrete and delay transitions as defined below.*

We now describe the two types of transitions in the CTS w.r.t. CTA as mentioned in Definition 4. In Definition 5 discrete transitions are described. When performing an enabled discrete transition, i.e. taking an edge where the guards on the edge evaluate to true and the invariant in the target location evaluates to true, the clocks in the reset set are set to zero and the target location becomes the active location.

Definition 5 (Discrete transitions). *A transition* $(l, v, q) \xrightarrow{a} (l', v', q)$ *is a discrete transition iff there is an edge* (l, gx, gc, a, r, l') *from* l *to* l'*, s.t.* $v, q \models gx \wedge gc$ *and* $v', q \models I(l')$ *and* v' *is derived from* v *by resetting all clocks in the reset set* r*.*

A delay transition is defined as time passing while remaining in the same location. We denote this delay by $d \in \mathbb{R}_{\geq 0}$. The definition of a delay transition is given in Definition 6.

Definition 6 (Delay transitions). *A transition* $(l, v, q) \xrightarrow{d} (l, v', q')$ *is a delay transition iff* $q' = q + d \cdot P(l)$, $v' = v + d$ *and the invariant for clocks and capacity variables in* l *is satisfied by the source, target and all intermediary states i.e.* $\forall 0 < d' \leq d$ *we have* $v + d', q + d' \cdot P(l) \models I(l)$.

A delay transition is only enabled when the invariant of the location holds both before, during and after the delay. The capacity variables are updated according to the product of cost rates in the location and the duration of the delay.

The transition between two locations l and l' with action a is denoted as $l \xrightarrow{a} l'$. For the impossible transition, we write $\overset{a}{\nrightarrow}$, that is a non-existing label is used or the invariant in the target location cannot be satisfied. We define a run ϱ as being an infinite sequence of transitions in the transition system of a CTA, of the form $\varrho = s_0 \xrightarrow{a} s_1 \xrightarrow{a} \cdots \xrightarrow{a} s_i \xrightarrow{a} \cdot$. We define π as a position in the run ϱ s.t. $\varrho[\pi]$ is the state, (l, v, q) at the given step in the run.

2.1 Parallel Composition

Concurrent execution and parallel composition is an integral part of not only TA but also SWA and LHA, and is also needed to model more complex capacity problems. In this section we define the semantics for parallel composition of CTA.

Definition 7 (Parallel Composition of Capacity Timed Automata). *Let* $PA = A_1|A_2$, *where* $A_i = (L_i, l_0^i, E_i, I_i, P_i)$ *for each* $i \in \{1, 2\}$, *be a parallel composition of two CTA over a finite set of clocks* X, *a finite set of real-valued capacity variables* C, *a finite set of initial values* O *for each capacity variable, a function* $B : C \longrightarrow \mathbb{Z} \times \mathbb{Z}$ *that assigns bounds for each capacity variable s.t.* $B(c_j) = [k_1, k_2]$, *where* $k_1, k_2 \in \mathbb{Z}$ *and a finite set of actions* $Act = \{\tau\} \cup N \cup \{c! \mid c \in \textbf{Chan}\} \cup \{c? \mid c \in \textbf{Chan}\}$, *where* **Chan** *is the finite set of channel names. We define the CTS* $\mathcal{T}(PA)$ *generated by the parallel composition of CTA PA as*

$$\mathcal{T}(PA) = (S, s_0, \Sigma, \longrightarrow)$$

where $S = \{(l_1, l_2, v, q) \mid (l_1, l_2, v, q) \in L_1 \times L_2 \times \mathbb{R}^X_{\geq 0} \times \mathbb{R}^C$ *and* $v, q \models \bigwedge_{i \in \{1,2\}} I_i(l_i)$ *and* $q(c_j) \models B(c_j)$ *for all* $c_j \in C\}$ *is a state where the clock and capacity variable valuations satisfy the invariants in location* l_1 *and* l_2 *and the capacity variable valuations satisfy their respective bounds.* $s_0 = (l_0^1, l_0^2, v_0, q_0)$ *is the initial state for* v_0 *evaluating to zero for all clocks in* X *and* $q_0(c_j) = o_j$ *for all* $c_j \in C$ *and* $o_j \in O$. $\Sigma = Act \cup \mathbb{R}_{\geq 0}$ *is the set of labels, and* $\longrightarrow$ *consists of delay, discrete and synchronizing transitions as defined below.*

Rule PA1 defines the case where the whole system performs a delay transition of d time units.

$$\text{PA1.}\ \frac{d \in \mathbb{R}_{\geq 0}}{\langle (l_1, l_2, v, q) \rangle \xrightarrow{d} \langle (l_1, l_2, v', q') \rangle}$$

where $v' = v + d$,
$q' = q + d \cdot (P_1(l_1) + P_2(l_2))$ and
$\forall 0 < d' \leq d,\ v + d', q + d' \cdot (P_1(l_1) + P_2(l_2)) \models I_1(l_1) \wedge I_2(l_2)$

PA1 states that the system can delay for d time units, where the clocks are increased by d and capacity variables are calculated, as long as the clock and capacity valuations satisfy the invariants of location l_1 and l_2 from the start of the delay to the end. The value of a capacity variable is changed according to the product of the delay and the sum of the capacity rates of all the locations, e.g. if a capacity represents a battery and one CTA is in a location that decreases power by 3 and another is in a location where the charge of the battery is increased by 5, the resulting rate is an increase of 2.

$$\text{PA2.}\ \frac{(l_1, gx, gc, a, r, l_1') \in E_1, v, q \models gx \wedge gc, v', q \models I_1(l_1') \wedge I_2(l_2)}{\langle (l_1, l_2, v, q) \rangle \xrightarrow{a} \langle (l_1', l_2, v', q) \rangle}$$

where v' is derived from v by resetting all clocks in the reset set r

$$\text{PA3.}\ \frac{(l_2, gx, gc, b, r, l_2') \in E_2, v, q \models gx \wedge gc, v', q \models I_2(l_2') \wedge I_1(l_1)}{\langle (l_1, l_2, v, q) \rangle \xrightarrow{b} \langle (l_1, l_2', v', q) \rangle}$$

where v' is derived from v by resetting all clocks in the reset set r

Rules PA2 and PA3 define the cases where the system can perform either the discrete action a or b and reach locations l_1' and l_2' respectively. PA2 shows how the system performs action a and reaches a new state iff there exists a transition from the source location l_1 to the target location l_1' and the clock and capacity variables satisfy the guards on the edge. The new clock valuations along with the capacity valuations must satisfy the invariant of the target location and location l_2.

$$\text{PA4.}\ \frac{(l_1, gx_1, gc_1, a!, r_1, l_1') \in E_1, (l_2, gx_2, gc_2, a?, r_2, l_2') \in E_2, v', q \models \varphi}{\langle (l_1, l_2, v, q) \rangle \xrightarrow{\tau} \langle (l_1', l_2', v', q) \rangle}$$

where v' is derived from v by resetting all clocks in $r_1 \cup r_2$ and
$\varphi = I_i(l_i') \wedge gx_i \wedge gc_i$ for each $i \in \{1, 2\}$

Rule PA4 defines how the system can perform a silent action τ and reach a new state. The system can perform this action iff both the automata running in parallel can perform an internal action where the clock and capacity valuations in the source locations in both automata satisfy the edge guards and the new clock valuation, given by resetting the clocks in the reset set, satisfy the invariants in both target locations. An example of synchronizing automata could be a robot and a charging station where the charging station performs an output action, *charge*!, over the channel *charge*. The robot that requires charging can synchronize with the charging station by performing an input action, *charge*?, over the channel *charge*.

3 Decidability of CTA

We now turn to a general property of CTA and propose the following theorem.

Theorem 1. *The reachability problem for CTA with one clock and two capacity variables is undecidable.*

To prove this theorem we use the notion of Minsky machines and the halting problem, which in the case of Minsky machines is proven undecidable [Hir94]. A Minsky machine is a two-counter machine with the two labelled commands; $L : c := c + 1$; *goto* L' and L : *if* $c = 0$ *then goto* L' *else* $c := c - 1$; *goto* L'' and the *HALT* command. By reducing the halting problem for Minsky machines to the reachability problem for CTA it can be concluded, through proof by contradiction, that the reachability problem for CTA is undecidable.

Proof. Assume the CTA A with two capacity variables c_1 and c_2, one clock x and the bounds $B(c_1) = B(c_2) = [0, 5]$. Let the two counters of the Minsky machine m_1 and m_2 be encoded by c_1 and c_2 as follows.

$$c_1 = 5 - \tfrac{1}{2^{m_1}},\ c_2 = 5 - \tfrac{1}{2^{m_2}}$$

For notational convenience we further denote $e_1 = \frac{1}{2^{m_1}}$ s.t. the capacity variable can be written $c_1 = 5 - e_1$. This also holds for $c_2 = 5 - e_2$ with $e_2 = \frac{1}{2^{m_2}}$.

Now, incrementing (resp. decrementing) either counter m_1 or m_2 is expressed by halving (resp. doubling) the value e. We introduce the following generic module $\mathbf{Mod}_{j,n}$ for handling counter increment and decrement as part of the commands of a Minsky machine.

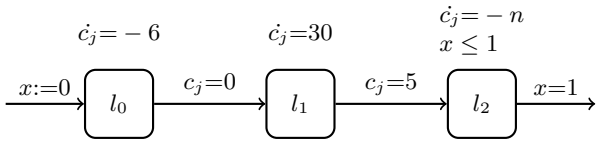


Figure 2. Module $\mathbf{Mod}_{j,n}$.

In location l_2 we let n be decided by the function of the module, as it is explained later. Also, with $\mathbf{Mod}_{1,n}$ we refer to a specific module which modifies c_1, and thus m_1. Capacity variable c_2 is disregarded in the module. That is, the cost rate $P(l)(c_2)$ is 0 in each location, and there are no invariants and guards in $\mathbf{\Delta}(c_2)$. This holds vice versa with $\mathbf{Mod}_{2,n}$ where c_1 is disregarded.

To further encourage how $\mathbf{Mod}_{j,n}$ handles the Minsky machine commands, consider the following example run through the module:

- the run enters location l_0 with the configuration $(l_0, x = 0, c_j = 5 - e)$
- it delays for $\frac{5-e}{6}$ in l_0, resulting in $(l_0, \frac{5-e}{6}, 0)$
- location l_1 is entered with the configuration $(l_1, \frac{5-e}{6}, 0)$
- it delays for $\frac{1}{6}$, resulting in $(l_1, 1 - \frac{e}{6}, 5)$
- entering the last location l_2, the configuration is $(l_2, 1 - \frac{e}{6}, 5)$
- the run is required to delay in l_2 until the guard $x = 1$ is satisfied. This results in $(l_2, 1, 5 - \frac{n \cdot e}{6})$

Leaving the last location l_2 it is seen that $c_j = 5 - \frac{n \cdot e}{6}$. Consequently, we now state the following about $\mathbf{Mod}_{j,n}$.

Lemma 1. *Any run entering* $\mathbf{Mod}_{j,n}$ *with* $c_j = 5 - e$ *will exit the module from* l_2 *s.t.* $c_j = 5 - \frac{n \cdot e}{6}$.

Proof. Lemma 1 is proved partly from the example run given above and partly from the forced forward progression in $\mathbf{Mod}_n$.

The example run shows the arithmetical manipulations on the capacity variable as a result of a run through the module, and given it's simplicity, no further elaboration is needed here. Note that invariants for the clock are only present in l_2 of the module. This is because we rely on the capacity guards and bound on c_j for forced forward progression. Consider $(l_0, \frac{5-e}{6}, 0)$ from the run above. If we were to delay beyond $\frac{5-e}{6}$ time units, it would result in $c_j < 0$ and thus the bound $B(c_j) = [0, 5]$ would be violated. From Definition 4 we see that this is not possible. Hence, we can conclude Lemma 1 to hold.

Now, assume the module $\mathbf{Mod}_{1,3}$ with $n = 3$ in l_2. A run through $\mathbf{Mod}_{1,3}$ will result in $c_1 = 5 - \frac{3e}{6}$. Effectively, the counter m_1 has been incremented by 1, because e has been halved.

Below, we will use the four modules $\mathbf{Mod}_{1,3}$, $\mathbf{Mod}_{2,3}$, $\mathbf{Mod}_{1,12}$ and $\mathbf{Mod}_{2,12}$ which will increment m_1, increment m_2, decrement m_1 and decrement m_2 respectively. As with the example of $\mathbf{Mod}_{1,3}$ above, similarly it is possible to prove

that the proposed values for n and j have the desired effect on the intended capacity variable.

Consider now the last module needed to simulate a Minsky machine, which is called $\mathbf{Test}_{j,n}$ and is depicted on Figure 3.

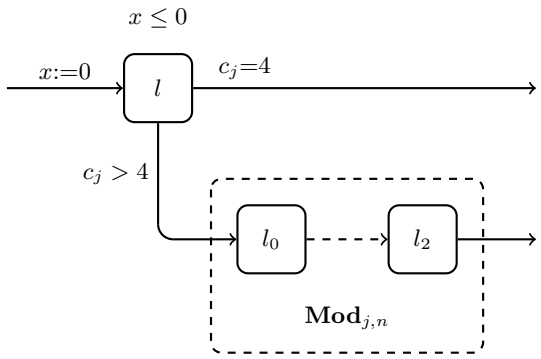


Figure 3. Module $\mathbf{Test}_{j,n}$.

A run entering $\mathbf{Test}_{j,n}$ will leave the location l via one of two possible edges, depending on the value of c_j. Effectively, the guards on these edges test either counter m_1 or m_2 (resp. module $\mathbf{Test}_{1,n}$ and $\mathbf{Test}_{2,n}$) and whether this counter is 0 or > 0. Only if the counter is > 0, thus $c_j > 4$, a run through the module continues through $\mathbf{Mod}_{j,n}$. Here, $\mathbf{Mod}_{1,12}$ is assumed if the module $\mathbf{Test}_{1,12}$ for c_1 is considered. With $\mathbf{Test}_{2,12}$ for c_2, $\mathbf{Mod}_{2,12}$ is assumed.

Lemma 2. *Any run entering a module* $\mathbf{Test}_{j,n}$ *where* $c_j = 5 - e$ *will leave the module from* l *immediately with* $c_j = 5 - e$ *if the counter,* m_1 *or* m_2*, considered is* 0*. If this counter is* > 0 *the run leaves the module with* $c_j = 5 - \frac{n \cdot e}{6}$

Proof. It follows immediately from the description above of $\mathbf{Test}_{j,n}$ and Lemma 1 on $\mathbf{Mod}_{j,n}$, that Lemma 2 holds, thus no further evidence is needed.

Now, with the CTA A and the two modules $\mathbf{Mod}_{j,n}$ and $\mathbf{Test}_{j,n}$ we can reduce the halting problem for Minsky machines to the reachability problem for CTA.

Let M be a Minsky machine. Initially the two counters m_1 and m_2 in M are 0, which in A means that the initial values for the two capacity variables o_1 and o_2 are $5 - \frac{1}{2^0} = 4$. Every increment command $L : c := c + 1$; *goto* L' in M we simulate with either $\mathbf{Mod}_{1,3}$ or $\mathbf{Mod}_{2,3}$ in A, depending on the counter being incremented. Likewise, every command $L : \mathit{if}\ c = 0\ \mathit{then\ goto}\ L'\ \mathit{else}\ c := c - 1$; *goto* L'' in M is simulated by $\mathbf{Test}_{1,12}$ or $\mathbf{Test}_{2,12}$ in A, depending on the counter. The *HALT* command we simply simulate in A with a single final location l_{HALT}.

It follows that M halts iff the location l_{HALT} is reachable in A. We can thus conclude the reachability problem for CTA with two capacity variables to be undecidable, because the halting problem for Minsky machines is undecidable.

4 Robotic Vacuum Cleaner

This section will present an example which can be encoded as a network of CTA. The example is a robotic vacuum cleaner, which cleans and returns to a charging station before it's battery runs low. Such a vacuum cleaner is supposed to run without the interaction of a human, except when the dustbin needs emptying. For the scope of this paper the models presented in this section are simplified models of a real robotic vacuum cleaner system.

The CTA will have two global capacity variables c_0 modelling the charge of the battery and c_1 modelling the dustbin. For all models $o_0 = 0$, $o_1 = 0$, $B(c_0) = [0, 10000]$ and $B(c_1) = [0, 100]$.

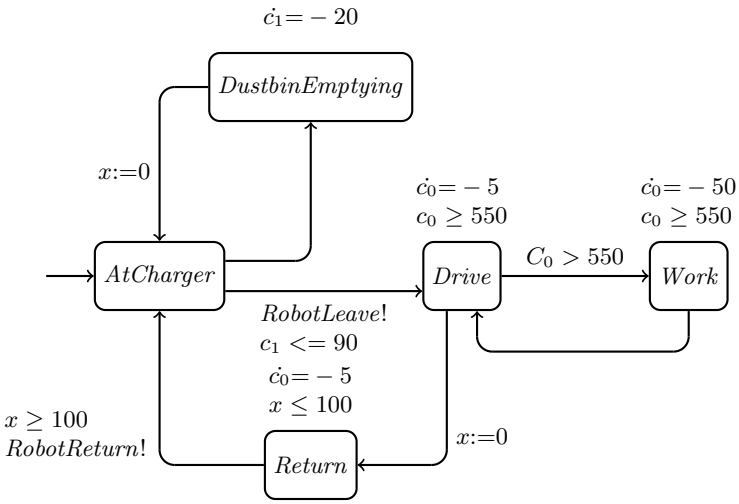


Figure 4. A CTA model of the robotic vacuum cleaner.

Figure 4 is a robotic vacuum cleaner modelled as a CTA. There are five locations; *AtCharger* is the robot being at the charger, *Drive* being the robot driving and *Return* is the robot returning to the charger. Locations *Drive* and *Return* both use five units of power per time unit. For simplicity it is assumed that the robotic vacuum cleaner is able to safely return to the charger within 100 time units. The last location *Work* is the robot driving and cleaning which uses 50 units of power while adding one unit of dust to the dustbin. The location *DustbinEmptying* is possible when the robot has connected to the charger. The location empties the dustbin, c_1, with a rate of -20 per time unit.

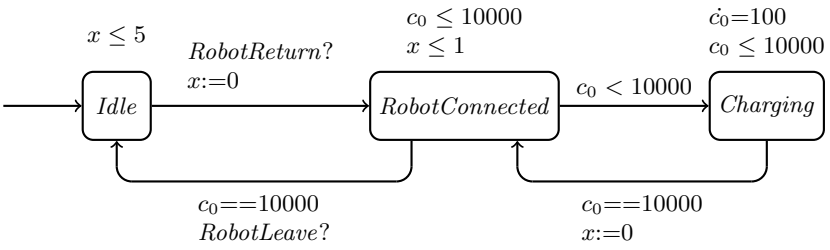


Figure 5. A CTA of the charging station for the robotic vacuum cleaner.

Figure 5 is a CTA of the charging station for the robotic vacuum cleaner. The model has a location *Idle* representing the robot not being at the charger, and thus the charger being idle. The location *RobotConnected* represents the robot being at the charger. For the sake of simplicity the CTA can only stay in this location for one time unit. In location *Charging*, which charges the robot, the robot must stay until the battery is fully charged. This is enforced by the invariant $c_0 \leq 10000$ in *Charging*.

For the robotic vacuum cleaner it would be interesting to verify whether it is possible for the system to reach a state where $c = 0$ and $x < 100$ while in location *Return*. If the system can reach such a state, it would halt, because time is not allowed to elapse further which would mean the robot would experience undesired behavior.

5 Expressiveness of CTA

With CTA we have defined a formalism similar to e.g. PTA or WTA, but with the possibility to model capacity problems. Figure 6 shows the hierarchy of the formalisms presented in Section 1.1.

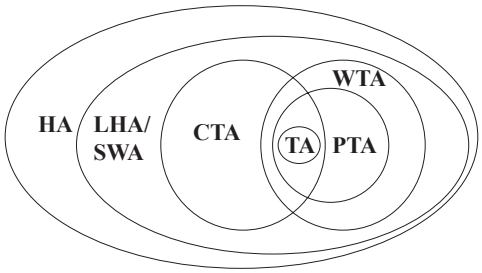


Figure 6. Expressiveness overview.

Gerd Behrmann *et al.* [BLR04] introduce PTA as an extension of the basic notion of TA. PTA extends the notion of TA with a price function on edges and locations that allows investigation of the accumulated price of a run in a given automata. The extension only allows a positive accumulated price to be found.

WTA are described as TA with a weight associated to each transition [BFL$^+$08]. This is quite close to the definition of PTA however the weight is defined as being any real valued number which allows transitions to have negative costs. This adds more expressiveness compared to PTA but MPTA [LR05] can simulate WTA using the difference between two price variables to express the weight of a single variable in a WTA.

Given the semantics of CTA it seems that CTA are capable of expressing more behaviour than WTA because it is allowed to have multiple real valued capacity variables. Though it remains an open problem whether it is possible to encode multiple capacity variables in a single weight of a WTA thus making CTA and WTA equally expressive with respect to capacity variables and their values. However the CTA formalism allows guards and invariants on capacity variables, a construct not available in WTA and PTA. This allows setting constraints on the locations and transitions of an automaton that limits the automaton with respect to the modelled capacity e.g. many intelligent chargers only charge the battery to some value and do not resume charging before the charge has dropped below another value. This is directly supported by CTA with the addition of global bounds on each capacity variable. Both WTA and PTA allow cost or weight associated with edges which is not possible in CTA. CTA is able to express behaviour that WTA/PTA is not and vice versa.

An open problem is investigating whether it is possible to simulate SWA using CTA making CTA equally expressive as SWA and LHA. We do not believe this to be the case. Our idea is that even though one can simulate a stopwatch, using a capacity variable and adjusting the rate to be either 0 or 1, it is not possible to become timed language equivalent as capacity variables cannot be reset like stopwatches can.

Franck Cassez *et al.* [CL00] prove two things; SWA and LHA are equally expressive and LHA are more restricted and thus less expressive than HA because only linear guards and invariants are allowed.

To further support our idea that CTA are not equal SWA (and thus LHA) in expressiveness, we give the example shown in Figure 7.

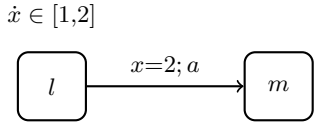


Figure 7. Example of an LHA that cannot be expressed as a CTA.

The example is based on the fact that LHA allow the rates to be defined as $Act \in ((\mathbb{Z} \times \mathbb{Z})^V)^N$ such that $Act(l)(x) = [u_1, u_2]$ defines that the rate of the variable x in location l lies in the compact bounded interval of $[u_1, u_2] \in \mathbb{Z} \times \mathbb{Z}$. This, as opposed to the definition of CTA where the rate must be a single integer, allows interval rates in locations which intuitively equals more behaviour.

6 Data Structure

This section will discuss a data structure which can be used to check reachability properties of CTA e.g. is it possible to reach a specific state and satisfy a clock and capacity variable property. We have proven in Section 3 that the reachability problem for CTA with one clock and two capacity variables is undecidable, and therefore we will need an over-approximating data structure, containing all the behaviour of the automaton. An over-approximation will contain all the correct behaviour of the automaton as well as additional behaviour. In Figure 8 this over-approximation is illustrated. We can see the correct behaviour **CTA** is contained in the over-approximated behaviour **CTA'** while the over-approximation is still smaller than all behaviour, Σ*****.

Also shown in Figure 8 is the relationship between the over-approximation and the type of queries we can do. If a query is found to hold true in the over-approximated behaviour we cannot know for certain if this is included in the real behaviour of the CTA, as illustrated by the point **b**. If we however find that some query does not hold for the over-approximation we can guarantee that it also does not hold for the real behaviour of the CTA as illustrated by the point **a**.

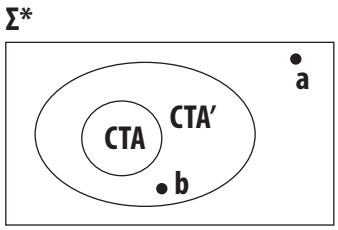


Figure 8. Over-approximation of behaviour.

CTS is defined such that the bounds of a capacity variable can not be violated, and therefore queries concerning capacity exceeding the bound is irrelevant. But still there are relevant queries with respect to capacity variables, e.g. the CTA in Figure 4 being in state $Work$ and $c_0 = 0$ would cause the vacuum cleaner to stop working and the system would come to a stop, which is an undesired behaviour.

6.1 Capacity Timed Computation Tree Logic

We do not have a way of formalizing queries on CTA at this point. We thus define *Capacity Timed Computation Tree Logic* (CTCTL) and provide a small example. CTCTL is a derivative of the *Weighted Computation Tree Logic* (WCTL), which is a logic that can be used to query PTA systems [BLM08]. As with WCTL and PTA systems, CTCTL can specify properties which can be queried for in a capacity timed finite state system. Specifically, CTCTL makes it possible to perform checks on the capacity variables and clocks in a system.

Definition 8 (Capacity Timed Computation Tree Logic). *A CTCTL formulae is defined inductively as:*

$$\phi := \textbf{\textit{true}} \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \exists\phi \; \mathcal{U}_{c\bowtie k}\phi \mid \forall\phi \; \mathcal{U}_{c\bowtie k}\phi$$

We let AP denote a set of atomic propositions and $p \in AP$, c is a capacity variable, $k \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$.

The CTCTL formulas possible by Definition 8 we interpret over labelled CTA. That is, we have a labelling function μ that, with every location l, associates a subset of AP.

The semantics of the satisfaction relation over (l, v, q), where l is a location, v is a clock valuation and q is a capacity valuation, we define as follows in Definition 9.

Definition 9 (Satisfaction Relation). *The CTCTL satisfaction relation over states (l, v, q) of labelled CTA A is defined inductively as:*

$$\begin{aligned}
(l, v, q) &\models \textbf{\textit{true}} \\
(l, v, q) &\models p \Leftrightarrow p \in \mu(l) \\
(l, v, q) &\models \neg\phi \Leftrightarrow (l, v, q) \not\models \phi \\
(l, v, q) &\models \phi_1 \wedge \phi_2 \Leftrightarrow (l, v, q) \models \phi_1 \text{ and } (l, v, q) \models \phi_2 \\
(l, v, q) &\models \exists\phi_1 \; \mathcal{U}_{c\bowtie k}\phi_2 \Leftrightarrow \text{there exists an infinite run } \varrho \text{ in } A \\
&\qquad \text{from } (l, v, q) \text{ s.t. } \varrho \models \phi_1 \; \mathcal{U}_{c\bowtie k}\phi_2 \\
(l, v, q) &\models \forall\phi_1 \; \mathcal{U}_{c\bowtie k}\phi_2 \Leftrightarrow \text{any infinite run } \varrho \text{ in } A \text{ from} \\
&\qquad (l, v, q) \text{ satisfies } \varrho \models \phi_1 \; \mathcal{U}_{c\bowtie k}\phi_2 \\
\varrho &\models \phi_1 \; \mathcal{U}_{c\bowtie k}\phi_2 \Leftrightarrow \text{there exists a position } \pi \text{ along } \varrho \\
&\qquad \text{s.t. } \varrho[\pi] \models \phi_2, \; \forall 0 > \pi' > \pi \text{ in} \\
&\qquad \varrho \text{ the position } \pi' \text{ requires } \varrho[\pi'] \models \\
&\qquad \phi_1, \text{ and } q(c) \bowtie k \text{ where } q \in \varrho[\pi]
\end{aligned}$$

In the following we will use the two abbreviations $\exists\Diamond_{c\bowtie k}\phi$ for $\exists\textbf{true} \; \mathcal{U}_{c\bowtie k}\phi$ and $\forall\Box_{c\bowtie k}\phi$ for $\forall\textbf{true} \; \mathcal{U}_{c\bowtie k}\phi$.

Using CTCTL we can query the model in Figure 4 for properties e.g. whether the model can reach a state where the location is $Work$ and $c_0 = 0$. The CTCTL query would look like:

$$\exists\Diamond_{c_0=0} \, Work \tag{1}$$

6.2 Difference Bound Matrix

For storing the time zones of a CTA we have chosen the data structure called Difference Bound Matrix (DBM) [Kat06]. DBMs have been used with TA [Kat06] and PTA [BLR04] and therefore we believe they can be used for CTA as well. A DBM recognizes that the difference between clocks in an automaton is constant, and can be represented in a $|X| + 1$ dimensional matrix The rows and columns are labelled with the clocks $x_i \in X \cup \{0\}$, 0 being the clock that is always 0, where rows are used for lower bounds on clock differences and columns are used for upper bound differences. The entries of the matrix c_{ij} are calculated as $x_i - x_j \prec c_{ij}$, where $\prec \in \{\leq, <\}$.

The entries of the matrix will then contain the difference between two clocks, the clock value, the negative clock value, zero or ∞, and in this way encodes the zones of a state system in a DBM.

The problem of modelling the capacity variables of a CTA as a clock in a time system, lies in the rate of the variables. The DBM depends on the difference between the encoded elements being constant, but the rate of the capacity variables in a CTA can have different rates, therefore the difference is not constant, and thus rendering the DBM unusable as described.

6.3 Initial Over-Approximation of Capacity Intervals

We propose an over-approximating approach to verify reachability properties in CTA. The over-approximation will give an absolute upper and lower bound of each capacity variable with respect to the time interval for each symbolic state consisting of the pair of a location and a DBM.

First we need to add an implicit clock called r which is used as a reference clock. This clock is reset to 0 on each edge thus always starts from 0. This allows for reading the minimum and maximum time that a CTA can stay in a location from the DBM.

Though it is impossible to model the capacity as a clock in a DBM it is possible to utilize it in other ways. With the above mentioned r clock it is possible to calculate the minimum and maximum values possible for the capacity variable in a given state. The minimum and maximum will be calculated as described in (2) and (3), respectively. In the equations we have denoted the minimum time of r as t_{min} and the maximum time of r as t_{max}.

$$c'_{min} = \begin{cases} c_{min} + t_{min} \cdot \dot{c} & if\ \dot{c} > 0 \\ c_{min} + t_{max} \cdot \dot{c} & if\ \dot{c} < 0 \\ c_{min} & if\ \dot{c} = 0 \end{cases} \tag{2}$$

$$c'_{max} = \begin{cases} c_{max} + t_{max} \cdot \dot{c} & if\ \dot{c} > 0 \\ c_{max} + t_{min} \cdot \dot{c} & if\ \dot{c} < 0 \\ c_{max} & if\ \dot{c} = 0 \end{cases} \tag{3}$$

The two functions c'_{min} and c'_{max} define an over-approximated interval in which each capacity variable is bounded. The over-approximated interval is defined in Definition 10.

Definition 10 (Capacity Intervals). *A capacity interval is either of the form:*

$$\zeta_1 c_{lower}, c_{upper} \zeta_2$$

or:

$$(c_{lower}, c_{upper}, \rho_1, \rho_2)$$

such that c_{lower} *is calculated using* c'_{min}, c_{upper} *is calculated using* c'_{max}, $\zeta_1, \zeta_2 \in \{[,]\}$ *and* $\rho_1, \rho_2 \in \{true, false\}$.

The two representations are interchangeable. Note that the value of $\rho_1 = true \Leftrightarrow \zeta_1 = [$ and $\rho_1 = false \Leftrightarrow \zeta_1 =]$. The truth values are reversed for ρ_2 and ζ_2.

Based on the operator of the constraints of r in the DBM we can calculate the value of ρ_1 and ρ_2 as:

$$\rho'_i = \begin{cases} \rho_i \wedge true & \textit{if operator of r } \in \{\leq, \geq\} \\ \rho_i \wedge false & \textit{if operator of r } \in \{<, >\} \end{cases} \tag{4}$$

With the DBM in place and the two above functions defined we can show how to calculate the over-approximated interval of a capacity variable in Algorithm 1.

Algorithm 1 Calculate Capacity Interval.

Require: $l, e, prev_{lower}, prev_{upper}$
1: Generate DBM for l
2: Apply guards of e on DBM
3: Calculate capacity interval for e based on $prev_{lower}, prev_{upper}$
4: Reset clocks in the reset set of e and the reference clock r in DBM

Algorithm 1 shows that we take as input a location l, the edge e we are calculating, and the lower and upper values of the capacity interval of l. Based on this we generate the DBM of the location l, apply the guards of e, calculate c_{min} and c_{max} using the two above functions and then reset the clocks in the reset set of e in the DBM.

The intervals are not only calculated for each edge but also for each location as the DBM of the location itself may have caused the CTA to halt as a result of reaching a state from which no further transitions are available and time is not allowed to progress.

To further encourage how this over-approximation works, consider the simplified vacuum robot example illustrated in Figure 9.

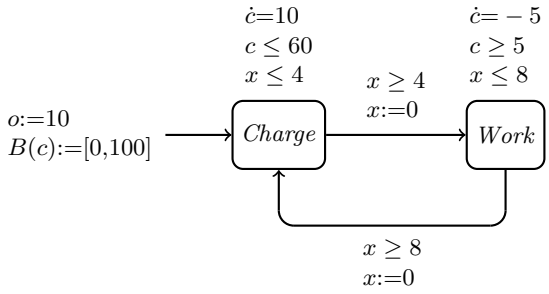
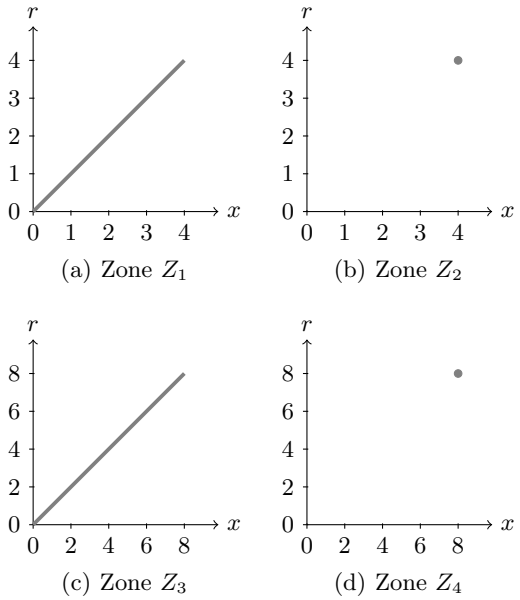


Figure 9. Simplified robot example.

Calculating the zone for each location and the zone passed to the next location yields the zones shown in Figure 10.



(a) Zone Z_1 (b) Zone Z_2

(c) Zone Z_3 (d) Zone Z_4

Figure 10. Zones for simplified robot example.

To illustrate how the computation of the capacity intervals is done we will look at a small example seen in (5). The bounds in the example are calculated for

the zone illustrated in Figure 10(a), where $\dot{c} = 10$, $c_{min} = c_{max} = 10$, $t_{min} = 0$ and $t_{max} = 4$, the maximum and minimum times can be read from the zone in Figure 10(a) as the two extremes of the r clock.

$$\begin{aligned} c'_{min} &= 10 + 0 \cdot 10 = 10 \\ c'_{max} &= 10 + 4 \cdot 10 = 50 \\ &\Rightarrow [10, 50] \end{aligned} \tag{5}$$

The symbolic trace below illustrates all of the intervals calculated for Figure 9:

$$\begin{aligned} (\mathit{Charge}, (Z_0, [10, 10])) \rightarrow (\mathit{Charge}, (Z_1, [10, 50])) \rightarrow (\mathit{Charge}, (Z_2, [50, 50])) \\ \rightarrow (\mathit{Work}, (Z_3, [10, 50])) \quad \rightarrow (\mathit{Work}, (Z_4, [10, 10])) \end{aligned} \tag{6}$$

Note that Z_0 is the initial zone where all clocks are zero and the capacity variables are at their initial values. We then construct the zone for location *Charge* and with that calculate the interval to be $[10, 50]$. Afterwards we limit the zone of *Charge* by applying the guard on the edge to *Work*. This will be the basis for the zone for *Work*. We calculate the interval for this edge's zone to be $[50, 50]$. We then proceed to do the same for *Work* giving the two next steps in the symbolic trace. The trace ends here because we now have a stable configuration. This is because on the edge from *Work* to *Charge* the x clock (and implicitly the r clock) is reset. This results in a zone that is exactly the same as the initial zone, thus we can write $Z_0 = Z_4$. We also have that the capacity interval of the edge is $[10, 10]$ which is contained in the initial interval of the symbolic trace thus we have seen all behaviour with respect to capacity values.

We can now perform reachability queries telling us whether the model has states that may be unsafe. In Table 1 some queries are shown along with their respective answer. With the first query we ask if a trace exists in which the automaton is in *Charge* and the capacity variable c is greater than or equal to 20. This is true, however it might be a result of the over-approximation and not the actual behaviour of the system. Thus nothing can be concluded on from this query. Consider instead the second query. Here we ask whether there exists a trace in which the automaton is in location *Work* and the capacity variable is less than 10. This is not possible and this answer also holds for the actual system and not just the over-approximation.

$\exists\Diamond_{c \geq 20}\, \mathit{Charge}$	$\exists\Diamond_{c < 10}\, \mathit{Work}$
True	False

Table 1. Example queries for Figure 10.

Note that the above example uses invariants and guards on the clock in each location to control progress through the model. However, in CTA we may also

rely on capacity invariants and guards for progress. In Section 6.5 we further elaborate on this with the example of Figure 11 which expands on Figure 9.

6.4 Deterministic Time CTA

From the example in Section 6.3 we can define Lemma 3.

Lemma 3. *Given any deterministic time CTA the over-approximated behaviour is exactly the real behaviour of the CTA.*

We clarify that deterministic time is when a transition between locations can occur at exactly a single point in time. Lemma 3 defines that the real behaviour of the CTA and the approximated is exactly the same. In Figure 8 this would mean **CTA** = **CTA'**.

Proof. We prove this by means of induction. Given that we enter a location with a capacity interval that contains exactly one value, that is e.g. $c_j \in [k_1, k_2]$ where $k_1 = k_2$, we will also exit the location with a capacity interval of exactly one value given that time is deterministic.

Base case: When a CTA is initialized the values of the capacity variables are initialized by the set O. The set O assigns a single value to the capacity variables, thus when we enter the initial location the interval of each capacity variable is exactly one value o_j.

Induction step: We assume that we enter a location with a capacity interval of a single point for each capacity variable. Given the assumption that time is deterministic we can see that $t_{min} = t_{max}$ as the minimum and maximum value of r is exactly a single point thus we can define $t = t_{min} = t_{max}$. Given the initial assumption that we enter the location with a capacity interval of a single value for each variable we can define $c_j = c^j_{min} = c^j_{max}$. If we insert this into the two functions used to calculate c'^j_{min} and c'^j_{max} we get a derived notion of the functions, because $t_{min} = t_{max}$:

$$c'^j_{min} = c^j_{min} + t \cdot \dot{c} \tag{7}$$
$$c'^j_{max} = c^j_{max} + t \cdot \dot{c} \tag{8}$$

If we insert c_j as well we see that:

$$c_j + t \cdot \dot{c} = c_j + t \cdot \dot{c} \tag{9}$$

This shows that given deterministic time and a capacity interval of a single value when we enter a location we will also exit the location with a capacity interval of a single value thus proving Lemma 3.

6.5 Refined Over-Approximation of Capacity Intervals

Figure 11 is an expanded model of Figure 9 and will serve as an example of how the initial over-approximation works with respect to capacity invariants and guards.

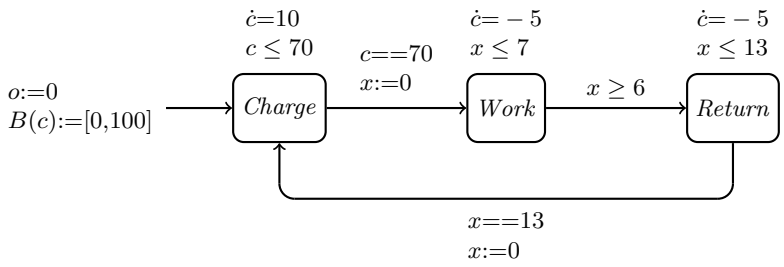


Figure 11. Extended robot example.

If we apply our over-approximation on this model a problem arises. The over-approximation is too large resulting in us not being able to properly verify the model. This can be seen in location *Charge*. There is no upper bound on the amount of time the automaton can be in this location thus the symbolic state will have the zone Z_1 as seen in Figure 12.

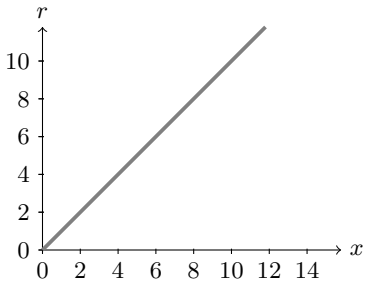


Figure 12. Zone Z_1 for location *Charge*.

With no upper bound on time, the calculation of c_{max} will result in the upper bound of the capacity interval being ∞.

Therefore, we need to refine the over-approximation to eliminate some behaviour thus minimizing the over-approximation. We add to Algorithm 1 a check that constrains the interval, if needed, by the capacity invariants (including the implicit invariants given by the bounds) or guards depending on whether it is an interval of a location or an edge that is being calculated. This adaptation can be seen in Algorithm 2.

Algorithm 2 Calculate Capacity Interval Refined.

Require: $l, e, prev_{lower}, prev_{upper}$
1: Generate DBM for l
2: Apply guards of e on DBM
3: Calculate capacity interval for e based on $prev_{lower}, prev_{upper}$
4: **if** invariant or guard is more strict than the current interval **then**
5: apply invariant or guard on interval
6: **end if**
7: Reset clocks in the reset set of e and the reference clock r in DBM

Applying the invariant or guard on the interval is done by exchanging the lower bound or upper bound by the constant of the invariant (or guard) depending on whether the invariant (or guard) defines a minimum or maximum value respectively. The inclusion of endpoints after applying the invariant (or guard) can be calculated as:

$$\rho_i' = \begin{cases} true & if \text{ operator of } \boldsymbol{\Delta}(c) \in \{\leq, \geq\} \\ false & if \text{ operator of } \boldsymbol{\Delta}(c) \in \{<, >\} \end{cases} \tag{10}$$

The location *Charge* has no upper limit on time but only on capacity, a common occurrence with electronic chargers which do not limit charging based on time but on capacity [Pan07]. The refinement adds a check to the two first lines of the above algorithm checking whether there is a more restrictive invariant (or guard when making intervals for edges) that can constrain the interval further. If a more constraining invariant (or guard) is found the value is used in the interval instead and the inclusion of end points is then based on the operator of the invariant (or guard).

$$\begin{aligned} (Charge, (Z_0, [0,0])) &\rightarrow (Charge, (Z_1, [0,70])) \rightarrow (Charge, (Z_2, [70,70])) \\ &\rightarrow (Work, (Z_3, [35,70])) \rightarrow (Work, (Z_4, [35,40])) \\ &\rightarrow (Return, (Z_5, [0,40])) \rightarrow (Return, (Z_6, [0,5])) \\ &\rightarrow (Charge, (Z_1, [0,70])) \end{aligned} \tag{11}$$

In (11) we see a symbolic trace of the automaton in Figure 11. All zones referred to in this trace can be seen in Appendix A.

We start in the initial zone in the initial location *Charge* with the capacity interval $[0, 0]$ given by the initial capacity value. As we discussed earlier on charging systems in general, no upper time bound can be set in this location thus we must with the new approach rely on the capacity invariants. In *Charge* two invariants are present; the implicit invariant $c \leq 100$ given by the bound and the invariant $c \leq 70$. The most strict invariant is the explicit invariant and it is thus used as the upper bound for the interval for c. The interval passed to *Work* is defined by the capacity guard given on the edge. This requires c to be exactly 70 giving an interval of $[70, 70]$. In *Work* and *Return* we calculate the interval just as we did with the example in Section 6.3 as there are no invariants that are more strict than those given by the zone.

Taking the edge back to *Charge* we return to charge with the same zone as when we started in *Charge* but with a different interval on the capacity variable. We thus need to recalculate the lower and upper bound of the interval for c. As we again limit the upper bound of c by the invariant in *Charge* we end up with the same interval $[0, 70]$ as previously and we have thus seen all behaviour of the over-approximation of the automaton.

As with the example from Section 6.3 we can now perform reachability queries on the automaton. Table 2 shows some examples. Again, we ask if there exists a trace in which the automaton is in *Charge* and the capacity variable is greater than or equal to 20 which is true. This leaves us with no certain information, because it might be part of the over-approximated behaviour. If we on the other hand ask the query whether there exists a trace in which the location is *Work* and the capacity variable is less than 35, we get the result false giving us a definitive answer that the automaton can never enter a state in which the capacity variable is less than 35 while being in *Work*.

$\exists\Diamond_{c>20}\, Charge$	$\exists\Diamond_{c<35}\, Work$
True	False

Table 2. Example queries for Figure 11.

6.6 Open Problems

An interesting problem occurs with the example in Figure 11 which we will be investigated in our future work. Consider the partial symbolic trace in (12) from the symbolic trace in (11).

$$\begin{aligned}(\mathit{Charge}, (Z_2, [70, 70])) \rightarrow (\mathit{Work}, (Z_3, [35, 70])) \rightarrow (\mathit{Work}, (Z_4, [35, 40])) \\ \rightarrow (\mathit{Return}, (Z_5, [0, 40])) \rightarrow (\mathit{Return}, (Z_6, [0, 5]))\end{aligned} \tag{12}$$

If we examine this trace and Figure 11 closer, we see that the minimum and maximum elapsed time between location *Work* and *Return* is 6 and 13 respectively. This is because the clock x is reset entering *Work*, limited by the guard $x \geq 6$ between *Work* and *Return* and limited by the invariant $x \leq 13$ in *Return*. As a result of this, the actual interval for c in *Return* is $[5, 40]$, which differs from our approximated interval of $[0, 40]$. The reason for this problem with our over-approximation, is that we do not capture the implicit relation between capacity intervals and cost rate of the previous location.

More explicitly, for the edge between *Work* and *Return* we have from (12) the symbolic state $(\mathit{Work}, (Z_4, [35, 40]))$. The interval $[35, 40]$ has the lower bound calculated according to the function in (2) with the assumption that the maximum of 7 time units have elapsed, as allowed per the invariant $x \leq 7$ in *Charge*. However, in the symbolic state $(\mathit{Return}, (Z_5, [0, 40]))$, for location *Return*, the

interval $[0, 40]$ is calculated from $[35, 40]$, where the lower bound of 0 is calculated by the function in (3) with the assumption that $t_{max} = 7$. As explained earlier, the previous lower bound of 35 was calculated under the assumption of 7 time units having elapsed in *Work*. Because a maximum of 13 time units are allowed to elapse over the two locations, t_{max} in *Return* should instead be 6. Effectively, with our over-approximation we assume a maximum of 14 time units allowed to elapse over *Return* and *Work*. This is because we do not capture the implicit relation between e.g. the interval $[35, 40]$ from the symbolic state $(Work, (Z_4, [35, 40]))$ and the cost rate in the previous location *Work* from which the interval was calculated.

In a worst case scenario it is clear that the interval and cost rate relation can depend recursively on all previous locations but we believe that it may be possible to get a sufficiently exact over-approximation by only looking at the previous location. We let this be an open problem for future work on improving the over-approximation.

7 Conclusion

In this paper we have refined the formalism of Capacity Timed Automata to address the defined problem domain of capacity problems. We use the notion of capacity variables to model varying values, such as the charge of a battery as shown in the robotic vacuum cleaner example. In addition to refining CTA we added parallel composition to the formalism such that it is possible to create networks of CTA.

We presented a proof that shows the reachability problem for CTA with one clock and two capacity variables to be undecidable. The proof is based on a reduction of the halting problem for Minsky Machines to the reachability problem for CTA.

The expressiveness of CTA in relation to other formalisms in the field of timed systems was discussed leading to an overview of the hierarchy of the formalisms.

As the reachability problem was proven undecidable we investigated the possibility of doing an over-approximation of the behaviour of a CTA to attempt approximate verification of reachability problems in CTA. We defined the notion of CTCTL, a variant of TCTL, which can be used to query the over-approximated model about capacity variables as well as clocks, resulting in true or false statements of properties of a CTA.

The data structure of Difference Bound Matrix was chosen to model time along with functions to calculate an interval in which capacity variables could be bounded in relation to time. We also refined the initial approximation, to allow more complex models to be verified, as it was too over-approximating meaning only very simple models could be verified.

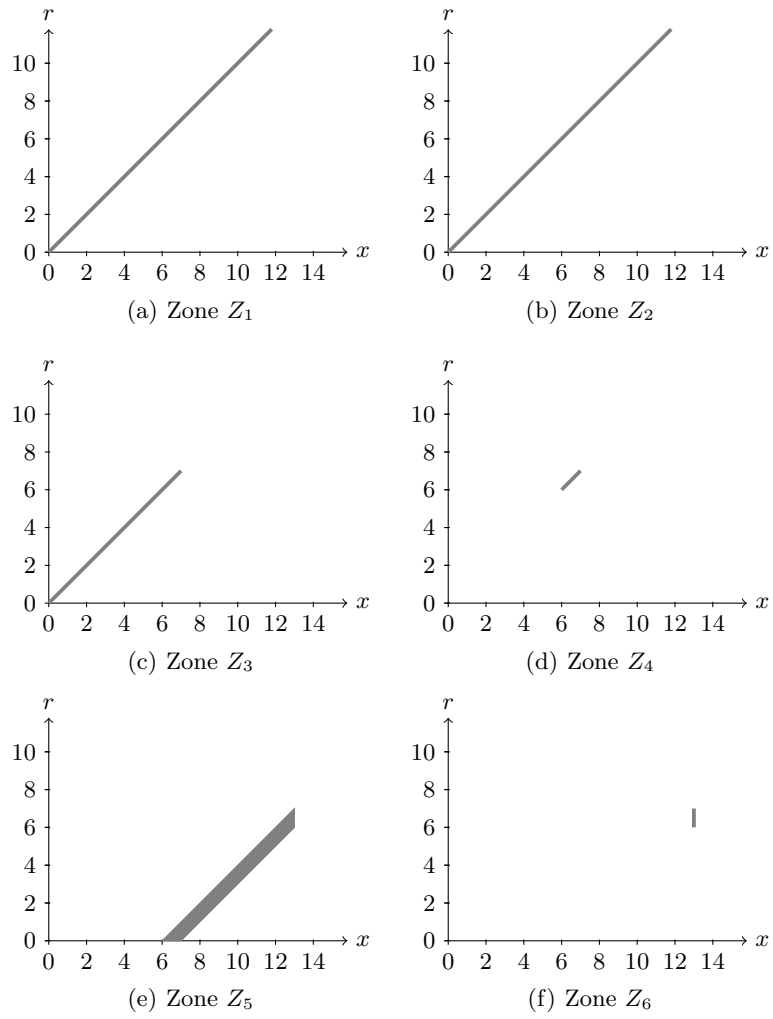
Future Work

The proposed over-approximating approach is still too over-approximating and needs to be further limited. As mentioned, a problem involving dependencies

between intervals and cost rates in locations, as discussed in Section 6.6, was discovered. It is clear that in a worst case scenario this relation is recursive over all previous locations however by limiting our view to only the previous location we believe we can get a sufficiently exact over-approximation. This is something we need to investigate in future work.

We also want to implement our over-approximation using the open source model checking framework *opaal* [opa10]. Opaal allows rapid prototyping of new model checking concepts such as our over-approximation.

Appendix A



(a) Zone Z_1 (b) Zone Z_2

(c) Zone Z_3 (d) Zone Z_4

(e) Zone Z_5 (f) Zone Z_6

Figure 13. Zones for the extended simplified robot example.

References

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:pp. 3–34, 1995.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, Vol. 126(Issue 2):pp. 183–235, 1994.

[AILS08] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiří Srba. *Reactive Systems - Modelling, Specification and Verification*. Number ISBN: 978-0-521-87546-2 in 1st edition. Cambridge University Press, 2008.

[BFL+08] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. *Lecture Notes in Computer Science*, 5215:pp. 33–47, 2008.

[BLM08] Patricia Bouyer, Kim Guldstrand Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. *CoRR*, abs/0805.1457, 2008.

[BLR04] Gerd Behrmann, Kim Guldstrand Larsen, and Jacob Illum Rasmussen. Priced timed automata: Algorithms and applications. *LNCS*, Vol. 3657:pp. 162–182, 2004.

[CL00] Franck Cassez and Kim Larsen. The impressive power of stopwatches. *LNCS*, Vol. 1877:pp. 138–152, 2000.

[COR10] UPPAAL CORA. Uppaal cora. http://www.cs.aau.dk/~behrmann/cora/, 2010. As seen December 4th, 2011.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. *IEEE Computer Society Press*, pages pp. 278–292, 1996.

[Hir94] Yoram Hirschfeld. Petri nets and the equivalence problem. *Lecture Notes in Computer Science*, 832:pp. 165–174, 1994.

[Kat06] Joost-Pieter Katoen. Zones and difference bound matrices. http://www-i2.informatik.rwth-aachen.de/i2/fileadmin/user_upload/documents/Advanced_Model_Checking/amc_lec18.pdf, 2006. As seen December 12th, 2011.

[LOHG10] Søren Larsen, Torkil Olsen, Anders Hesselager, and Jonas Groth. Capacity timed automata. May 2010. https://services.cs.aau.dk/public/tools/library/files/rapbibfiles1/1274991178.pdf.

[LR05] Kim Larsen and Jacob Rasmussen. Optimal conditional reachability for multi-priced timed automata. In Vladimiro Sassone, editor, *Foundations of Software Science and Computational Structures*, volume 3441 of *Lecture Notes in Computer Science*, pages 234–249. Springer Berlin / Heidelberg, 2005.

[opa10] opaal. opaal. http://opaal-modelchecker.com/, 2010. As seen December 13th, 2011.

[Pan07] Panasonic. Overcharge / overdischarge / overcurrent safety circuits. http://industrial.panasonic.com/www-data/pdf/ACA4000/ACA4000PE4.pdf, 2007. As seen December 4th, 2011.

[UPP10] UPPAAL. Uppaal. http://www.uppaal.com/, 2010. As seen December 4th, 2011.