On the Undecidability of Universality for Timed Automata with Minimal Resources

Sara E. Adams
sara.adams@exeter.oxon.org

Supervisor: Joël Ouaknine

Trinity 2006

Submitted in partial fulfilment of the requirements for the

Degree of Master of Science
in
Mathematics and the Foundations of Computer Science
at the
University of Oxford

Acknowledgements

I am grateful to Joël Ouaknine for introducing me to a most interesting area and for being a great supervisor throughout the dissertation. Without his support and our enlightening discussions this thesis would not have been possible.

I thank Michael Collins for guiding me through the programme "Mathematics and the Foundations of Computer Science" and for never being tired of answering my various questions on the programme.

Ulf Ochsenfahrt deserves my gratitude for supporting me in various ways; sharing my enthusiasm when things worked out, enduring my bad mood when they did not and last but not least giving me helpful feedback when writing up my results. I also thank Stefan Goldmann, Christian Hundt, Christian Kotitschke and Stephanie Tien Lee for helping me polish the thesis.

Finally I am appreciative of the Studienstiftung des deutschen Volkes (German National Academic Foundation) and Allianz for helping me finance my studies abroad.

Abstract

In 1994 Alur and Dill introduced timed automata and showed that universality was undecidable there. Since then it has been shown that under certain restrictions the problem becomes decidable. But the frontier between decidability and undecidability is still vast.

We aim at narrowing this gap considerably. Our main accomplishment is to prove that universality stays undecidable over weakly monotonic time when restricting to a single state, a single symbol and clock comparisons to 0 and 1 only. We further propose that over strongly monotonic time the problem stays undecidable for timed automata with one state and one symbol only.

Contents

Chapter 1

Introduction

In this first chapter we give a short history of timed automata and motivate our work. Thereafter we give an outline of this document to give a general idea of our proceeding.

1.1 Motivation

Systems can be modelled with finite state machines such as Kripke structures or finite automata. Unfortunately time cannot be modelled explicitly within these frameworks: while we can express the relative order of events, we cannot add specific time constraints. Hence timed automata are the preferred framework for real-time projects.

A timed automaton is simply a finite automaton that is extended by a time component. Clocks are introduced to keep track of time. Transitions are then restricted to instances where the clock values satisfy a given boolean formula of comparisons to constants. Furthermore clocks may be reset after each transition.

Timed automata can be used for model checking properties given in real-time temporal logics [1, 2]. One prominent example is the test series for Bang & Olufsen, in which a hitherto intractable error was located [3]. Another important application is found in conformance testing [4], where the goal is to determine whether a given implementation conforms to its specification.

Both instances are highly dependent on language inclusion. However, Alur and Dill showed that this problem is undecidable for timed automata [5]. Hence subclasses and variations of timed automata were examined extensively [6, 7, 8, 9, 10, 11]. It was proved that language inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ is decidable if $\mathcal{B}$ only has one clock or if it only uses clock comparisons to 0. The problem is also decidable if $\mathcal{B}$ is an open timed automaton, $\mathcal{A}$ is a closed timed automaton and weakly monotonic time is assumed.

However, it is equally important to determine when undecidability is reached. There is still a vast frontier between decidability and undecidability of language inclusion. We narrow this gap significantly by proving that language inclusion stays undecidable over weakly monotonic time when restricting virtually everything but the number of clocks. Weakly monotonic time refers to a model of time where simultaneous events are allowed. By contrast, in strictly monotonic time at most one event may occur at a given time.

More precisely, we show that universality — which is a special case of language inclusion — is undecidable over weakly monotonic time when only allowing clock comparisons to 0 and 1, a single state and a single symbol in the alphabet. Indeed this is a somewhat surprising result. When restricting both states and symbols all data has to be stored in the clocks. But a data structure that is dependent on clocks is very difficult to handle: the only permitted manipulation is resetting clocks to

0; information retrieval is restricted to comparing clocks to integers; and finally the information is continuously altered as the clock values are increased with the passage of time. Hence careful reasoning is necessary.

Furthermore, we alter the result to strongly monotonic time by proposing that universality stays undecidable when restricting to a single state and a single symbol. Note that in neither case do we allow diagonal constraints, i.e. comparing two clocks to each other.

1.2 Outline

In Chapter 2 we first define finite automata and then extend these to timed automata. After giving a simple example we alter the definition of timed automata slightly and prove that this does not change the expressive power. Next we introduce 2-counter machines in Chapter 3 and give a method of encoding their computations in timed words. This encoding is then used in Chapter 4 to give a new proof for the known result that the Universality Problem for timed automata is undecidable when allowing at most 2 clocks and clock comparisons to the constants 0 and 1 only. In particular, for that proof we only use specific timed automata, which we will call flat timed automata in Chapter 5. After converting flat timed automata to timed automata with one state and one symbol only we conclude our main new result: the undecidability of timed automata over weakly monotonic time with only one state and one symbol using no more than the constants 0 and 1. We further give a proof idea that extends a weaker version of our result to strongly monotonic time. In Chapter 6 we finalise our work where we sum up our results, mention problems that occurred in the process and point out possible future work.

Chapter 2

Timed Automata

Timed automata, which were introduced by Alur and Dill in [5], can be seen as an extension of finite automata. Hence in the following we give a quick definition and an example of finite automata to introduce our naming conventions. Both definition and example are then extended to timed automata by introducing a model of time. This approach is to ease the understanding of timed automata.

2.1 Finite Automata

Definition 2.1 A *finite automaton* $\mathcal{A}$ is a 5-tuple $\langle Q, \Sigma, q_0, \Delta, F \rangle$ where

- Q is a finite set of states
- Σ is a finite alphabet
- $q_0 \in Q$ is the initial state
- $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation and
- $F \subseteq Q$ is the set of final states. $\star$

We next define the language $\mathcal{L} \subseteq \Sigma^*$ accepted by a finite automaton. For this we introduce the notion of a computation, which is later extended to computations for timed automata. We will denote states with a q or an s: q represents a state of an automaton whereas s represents a state in a computation of an automaton. The main difference is that while every state has a unique name, in a computation you can visit a state more than once. Hence, when indexing, it can occur that $s_i = s_j$ for $i \neq j$ whereas we require that $q_i \neq q_j$ for $i \neq j$.

Definition 2.2 An *accepting computation* of a finite automaton $\mathcal{A} = \langle Q, \Sigma, q_0, \Delta, F \rangle$ is a finite sequence

$$(s_1, \sigma_1), (s_2, \sigma_2), \ldots, (s_k, \sigma_k)$$

such that $s_i \in Q$, $\sigma_i \in \Sigma$ and the following properties hold:

- it is initial, i.e. $s_1 = q_0$
- it is consecutive, i.e. $\forall 1 \leq i < k : (s_i, \sigma_i, s_{i+1}) \in \Delta$
- it is halting, i.e. $\exists s_f \in F : (s_k, \sigma_k, s_f) \in \Delta$

We sometimes write

$$(s_1, \sigma_1), (s_2, \sigma_2), \ldots, (s_k, \sigma_k), s_f$$

to indicate which final state was reached. $\star$

Definition 2.3 The *language accepted by* $\mathcal{A}$ is the set of all words $\sigma = \sigma_1 \ldots \sigma_k \in \Sigma^*$ such that there exists an accepting computation $(s_1, \sigma_1)(s_2, \sigma_2) \ldots (s_k, \sigma_k)$. The empty word λ is accepted iff $q_0 \in F$. $\star$

Example 2.4 Let

$$\mathcal{A} = \langle \{q_0, q_1\}, \{a, b\}, q_0, \{(q_0, a, q_1), (q_1, b, q_0)\}, \{q_0\} \rangle$$

be a finite automaton. Figure 2.1 shows the state graph $G_{\mathcal{A}}$ of $\mathcal{A}$: every state is circled, final states are double-circled; a transition (q, a, q') is represented by a directed edge from q to q' that is labelled with a. The initial state is marked by an edge that does not start in a state.

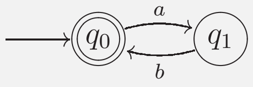

Figure 2.1: Example of a finite automaton

A word w is accepted by $\mathcal{A}$ iff there exists an *accepting walk* $(s_1, s_2, ..., s_k) \in G_{\mathcal{A}}$ such that $s_1 = q_0$, $s_k \in F$ and the edge labels assemble w.

For example, let $w = ababab$. Then the sequence

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_0$$

is an *accepting walk* in $\mathcal{A}$. We start in the initial state q_0 and stop in a final state q_0. Additionally, the labelling of the edges exactly forms the word *ababab*.

Observe that the language accepted by $\mathcal{A}$ is

$$L(\mathcal{A}) = \{(ab)^n : n \in \mathbb{N}_0\} = (ab)^*.$$

$\star$

2.2 Introducing Time

We now introduce time into our model of words and finite automata. For words the basic idea is to log when every letter occurs, i.e. each letter is assigned a non-negative real value. Note that over *weakly monotonic time* we allow subsequent letters to be assigned the same time, but we must not travel back in time, i.e. the sequence of times is nondecreasing. In contrast over *strongly monotonic time* simultaneous letters are not allowed, either. Furthermore, we call a letter an *event* to emphasise that it occurs at a specific time.

Definition 2.5 A *weakly timed word* over a finite alphabet Σ is a pair (σ, τ) where $\sigma = \sigma_1\sigma_2 \ldots \sigma_k \in \Sigma^k$ and $\tau = \tau_1 \ldots \tau_k \in (\mathbb{R}_0^+)^k$ is a nondecreasing sequence of real-valued time. A *strongly timed word* over a finite alphabet Σ is a pair (σ, τ) where $\sigma = \sigma_1\sigma_2 \ldots \sigma_k \in \Sigma^k$ and $\tau = \tau_1 \ldots \tau_k \in (\mathbb{R}_0^+)^k$ is a strictly increasing sequence of real-valued time. We often write $(\sigma_1, \tau_1)(\sigma_2, \tau_2) \ldots (\sigma_k, \tau_k)$ for (σ, τ). $\star$

In the following we often mention timed words without specifying whether they are weakly or strongly timed. Note that if weakly monotonic time is assumed, then the word has to be weakly timed. Similarly, if strongly monotonic time is assumed the word has to be strongly timed. If there is no restriction on the model of time, then both instances are possible.

Next we extend the concept of finite automata so that they include time. For this we assume that when the automaton starts running we are at time 0. Note that some time may pass before the first event occurs. We additionally allow a finite number of clocks: these can be compared to rational values and reset to 0 on any event. It is important that all clocks be incremented synchronously, i.e. if one time unit passes all clocks are incremented by 1.

Transitions for finite automata were only dependent on states and letters. For timed automata we add two components: first we may restrict a transition by only allowing instances where the clocks satisfy some boolean formula of comparisons; second we may reset some of the clocks on a transition. Note that the restriction has to be satisfied first and the resets are performed afterwards.

Definition 2.6 A *timed automaton* is a 6-tuple $\langle Q, \Sigma, C, q_0, \Delta, F\rangle$ where

- Q is a finite set of states,
- Σ is a finite alphabet,
- C is a set of real-valued variables, called clocks, that are synchronously incremented with the passage of time,
- $q_0 \in Q$ is the initial state,
- $\Delta \subset Q \times \Sigma \times \Phi(C) \times \mathcal{P}(C) \times Q$ is a finite transition relation and
- $F \subseteq Q$ is the set of final states.

Here $\mathcal{P}(C)$ denotes the power set of C and $\Phi(C)$ is the set of all clock constraints. These are inductively defined by

$$\delta := \texttt{true} | x \sim c | \delta \wedge \delta | \delta \vee \delta$$

where $x \in C$, $c \in \mathbb{N}_0$ and $\sim \in \{<, >, \leq, \geq, =, \neq\}$. ⋆

As seen with finite automata we next define the language that is accepted by a timed automaton. Speaking simply it is the set of all timed words such that the untimed word is accepted by the implicit finite automaton. Additionally, a transition can only be taken if the clock constraints are met and we complete every transition by resetting the clocks $r_C \subseteq \mathcal{P}(C)$ accordingly.

Definition 2.7 Let $\mathcal{A} = \langle Q, \Sigma, C, q_0, \Delta, F \rangle$ be a timed automaton where the set of clocks is $C = \{x_1, x_2, \ldots, x_k\}$. An *accepting computation* of $\mathcal{A}$ is a finite sequence

$$(s_1, \sigma_1, \tau_1), (s_2, \sigma_2, \tau_2), \ldots, (s_n, \sigma_n, \tau_n)$$

where $s_i \in Q, \sigma_i \in \Sigma$ and $\tau_i \in \mathbb{R}^+, \tau_i \leq \tau_{i+1}$ such that it is initialised, consecutive and halting. The computation is consecutive if there exist k sequences of times

$$t_{l,1},\ t'_{l,1},\ t_{l,2},\ t'_{l,2},\ \ldots,\ t_{l,n},\ t'_{l,n}$$

for $1 \leq l \leq k$ that represent the values of the clocks x_l. Furthermore, the sequences have to satisfy the following properties for all $1 \leq i < n$:

- $\exists (s_i, \sigma_i, \phi_i, r_i, s_{i+1}) \in \Delta$
- $(t_{1,i}, t_{2,i}, \ldots, t_{k,i}) \models \phi_i$
- $t'_{j,i} = \begin{cases} 0 & x_j \in r_i \\ t_{j,i} & \text{otherwise} \end{cases}$
- $t_{j,i+1} = t'_{j,i} + (\tau_{i+1} - \tau_i)$

For initiality we require that $s_1 = q_0$ and $\forall 1 \leq l \leq k : t_{l,1} = \tau_1$. A computation is halting if $\exists s_f \in F : (s_n, \sigma_n, \phi_n, r_n, s_f) \in \Delta$ where $(t_{1,n}, t_{2,n}, \ldots, t_{k,n}) \models \phi_n$ and $r_n \subseteq \mathcal{P}(C)$ is arbitrary.

We sometimes write

$$(s_1, \sigma_1, \tau_1), (s_2, \sigma_2, \tau_2), \ldots, (s_{n-1}, \sigma_{n-1}, \tau_{n-1}), (s_n, \sigma_n, \tau_n), s_{n+1}$$

to indicate which final state was reached. $\star$

Remark In the sequence of times $t_{l,i}$ represents the value of the clock x_l when checking the restriction $\phi_i \in \Phi(C)$ whereas $t'_{l,i}$ represents the value of x_l after the resets r_i have been performed. For consecutiveness we essentially check that transitions exist such that at every event the clock constraint ϕ_i is met. If ϕ_i was met, then clocks $x \in r_i$ are reset. Furthermore all clocks will have been incremented by exactly $\tau_{i+1} - \tau_i$ at the time of the next event.

Note that we can determine the sequences of times given a (possible) computation. First find all sequences of transitions such that each of them matches the given state changes and occurring events. Then go through each of these sequences linearly and keep track of which values all clocks have. The transitions tell us which clocks to reset and the computation shows how much time passes between each event. Finally check if all clock constraints are met to guarantee consecutiveness. $\star$

Now we can define the language accepted by a timed automaton. Observe that timed automata are nondeterministic.

Definition 2.8 The *language accepted* by a timed automaton $\mathcal{A} = \langle Q, \Sigma, C, q_0, \Delta, F \rangle$ is the set of all timed words (σ, τ) such that there exists an accepting computation for (σ, τ). Let $\sigma = \sigma_1, \ldots, \sigma_n$ and $\tau = \tau_1, \ldots, \tau_n$. For the existence of an accepting computation there has to exist a sequence $s_1, \ldots, s_n$ where $s_i \in Q$ such that $(s_1, \sigma_1, \tau_1), \ldots, (s_n, \sigma_n, \tau_n)$ is an accepting computation. $\star$

Example 2.9 Let $\mathcal{A} = \langle \{q_0, q_1\}, \{a, b\}, \{x\}, q_0, \Delta, \{q_0\}\rangle$ where

$$\Delta = \big\{\big(q_0, a, \{x = 1\}, \{x\}, q_1\big), \big(q_1, b, \{x = 1\}, \{x\}, q_0\big)\big\}$$

Figure 2.2 shows the state graph $G_{\mathcal{A}}$ of $\mathcal{A}$. Conventions are as for finite automata,

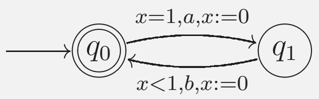

Figure 2.2: Example of a timed automaton

but the edge labelling is expanded by the clock constraints and the clocks to be reset. Note that the symbol '=' indicates an equality-check whereas a ':=' indicates a reset of a clock.

Observe that the automaton described is an extension of Example 2.4, but the clock constraints ensure that for any word accepted by the automaton the first event 'a' is at time 1, after each 'a' the next event 'b' occurs before a full time unit passes and after each 'b' the next event 'a' occurs after exactly one full time unit passed. More formally the language accepted by $\mathcal{A}$ is

$$\begin{aligned}\mathcal{L}(\mathcal{A}) = \{(\sigma,\tau) :\ & \sigma \in (ab)^*, \tau_1 = 1,\\ & \forall 1 \le i \le \tfrac{|\sigma|}{2} : \tau_{2i} - \tau_{2i-1} < 1,\\ & \forall 1 \le i < \tfrac{|\sigma|}{2} : \tau_{2i+1} - \tau_{2i} = 1\}.\end{aligned}$$

⋆

Remark A state graph of a timed automaton $\mathcal{A}$ as seen above gives a full description of $\mathcal{A}$, i.e. a full formal definition can be extracted from it.

In the following we usually give a state graph instead of a formal definition. Then the labelling of the states can be omitted. On transition edges we also allow sets of events instead of single events only. These edges intuitively express that any event in the set is allowed. This corresponds to a set of edges which all have the same restrictions and resets, but only a single event. See Figure 2.3 for an example. $\star$

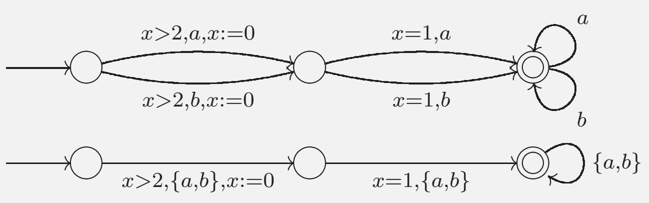

Figure 2.3: Omitting state labelling and allowing sets of events

Instead of one initial state in the following we will allow a set of initial states for timed automata and show that this does not change the expressive power. However, it does ease some operations on timed automata. For example, when allowing multiple initial states, the union of timed automata is very straightforward:

Let $\mathcal{A}_1 = \langle Q_1, \Sigma_1, C_1, I_1, \Delta_1, F_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, \Sigma_2, C_2, I_2, \Delta_2, F_2 \rangle$ be timed automata. Then the automaton

$$\mathcal{A} = \langle Q_1 + Q_2, \Sigma_1 \cup \Sigma_2, C_1 + C_2, I_1 + I_2, \Delta_1 + \Delta_2, F_1 + F_2 \rangle$$

accepts the language $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. We use '+' for the disjunct union $\dot\cup$ to stress that we do not use it for the alphabet. In the other instances renaming might be necessary.

Definition 2.10 (alternate definition) A *timed automaton* is a 6-tuple

$$\mathcal{A} = \langle Q, \Sigma, C, I, \Delta, F \rangle$$

where I is a set of initial states and Q, Σ, C, Δ, F are as in Definition 2.6.

For initiality of a computation $(s_1, \sigma_1, \tau_1), (s_2, \sigma_2, \tau_2), \ldots$ we now only require $s_1 \in I$ instead of $s_1 = q_0$. $\star$

Definition 2.11 A timed automaton $\mathcal{A} = \langle Q, \Sigma, C, I, \Delta, F \rangle$ is *weakly timed universal* iff it accepts every weakly timed word over Σ. It is *strongly timed universal* iff it accepts every strongly timed word over Σ. $\star$

Lemma 2.12 *Definition 2.6 and Definition 2.10 are equivalent, i.e. for any automaton complying to one definition there exists an automaton complying to the other definition and both accept the same language.*

Proof. '2.6⇒2.10': Given a timed automaton $\mathcal{A} = \langle Q, \Sigma, C, q_0, \Delta, F \rangle$ the timed automaton $\mathcal{A}' = \langle Q, \Sigma, C, \{q_0\}, \Delta, F \rangle$ accepts the same language as $\mathcal{A}$.

'2.10⇒2.6': Given a timed automaton $\mathcal{A} = \langle Q, \Sigma, C, I, \Delta, F \rangle$ with $q_0 \notin Q$ we construct a timed automaton $\mathcal{A}' = \langle Q', \Sigma, C, q_0, \Delta', F' \rangle$ as follows:

- $Q' = Q \,\dot{\cup} \{q_0\}$
- $\Delta' = \Delta \cup \{(q_0, \sigma, \phi, r, s) | \exists s_i \in I : (s_i, \sigma, \phi, r, s) \in \Delta\}$
- $F' = \begin{cases} F & I \cap F = \emptyset \\ F \cup \{q_0\} & \text{otherwise} \end{cases}$

Then $\mathcal{A}'$ accepts the same language as $\mathcal{A}$. $\square$

Chapter 3

2-Counter Machines

Although we determined which language is accepted by the automaton in Example 2.2, in general this is a hard problem. In Chapter 4 we will show that even deciding whether a timed automaton is universal, i.e. whether the automaton accepts all timed words, is undecidable. For the proof we will reduce the Halting Problem for 2-counter machines to the Universality Problem for timed automata. This chapter first introduces 2-counter machines and then describes a time-encoding of computations, an important tool in our proofs.

3.1 Basics

A *2-counter machine* consists of a finite state machine with two attached counters that are initially zero. On every transition each counter is either incremented or decremented. Additionally, we can check whether a counter is zero and use this as a restriction on transitions.

Definition 3.1 A *2-counter machine* is a 4-tuple $\langle Q, q_0, \Delta, q_f \rangle$ where

- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\Delta \subseteq Q' \times R \times \{I, D\}^2 \times Q$ where $Q' = Q \backslash \{q_f\}$ is the transition relation and
- $q_f \in Q \backslash \{q_0\}$ is the final state.

The set $\{I, D\}$ is used for incrementing or decrementing each counter and

$$R = \mathcal{P}(\{c = 0, d = 0\}) = \big\{\emptyset, \{c = 0\}, \{d = 0\}, \{c = 0, d = 0\}\big\}$$

determines the restrictions for each transition. A transition is restricted to instances where the first counter is 0 if it includes '$c = 0$'. Similarly, it is restricted to instances where the second counter is 0 if it includes '$d = 0$'. ⋆

Definition 3.2 A *configuration* of a 2-counter machine $\mathcal{M} = \langle Q, q_0, \Delta, q_f \rangle$ is a 3-tuple (s, c, d) where $s \in Q$ is the current state, $c \in \mathbb{N}_0$ is the value of the first counter and $d \in \mathbb{N}_0$ is the value of the second counter.

A *successor configuration* (s', c', d') of a configuration (s, c, d) of $\mathcal{M} = \langle Q, q_0, \Delta, q_f \rangle$ is a configuration of $\mathcal{M}$ such that $c' \in \{c - 1, c + 1\}, d' \in \{d - 1, d + 1\}$ and $\exists (s, r, a_c, a_d, s') \in \Delta$ where

- '$c = 0$' $\notin r$ if $c > 0$ and '$d = 0$' $\notin r$ if $d > 0$
- $a_c = \begin{cases} I & c' = c + 1 \\ D & c' = c - 1' \end{cases}$ and $a_d = \begin{cases} I & d' = d + 1 \\ D & d' = d - 1 \end{cases}$

⋆

Remark Note that a counter cannot be decremented if it is zero. So a transition that decrements a counter can only be taken if the counter has a positive value in its current state. In other words, any transition $(\cdot, r, D, \cdot, \cdot)$ can be interpreted as a transition $(\cdot, r \cup \{c \neq 0\}, D, \cdot, \cdot)$ and similarly $(\cdot, r, \cdot, D, \cdot)$ as $(\cdot, r \cup \{d \neq 0\}, \cdot, D, \cdot)$. $\star$

Definition 3.3 A *halting computation* of a 2-counter machine $\mathcal{M} = \langle Q, q_0, \Delta, q_f \rangle$ is a finite sequence

$$(s_1, c_1, d_1)(s_2, c_2, d_2) \dots (s_k, c_k, d_k)$$

such that

- it is initial, i.e. $(s_1, c_1, d_1) = (q_0, 0, 0)$,
- it is consecutive, i.e. $\forall 1 \leq i < k : (s_{i+1}, c_{i+1}, d_{i+1})$ is a successor configuration of (s_i, c_i, d_i)
- and it is halting, i.e. $s_k = q_f$ is the final state of $\mathcal{M}$.

$\star$

Example 3.4 Let $\mathcal{M} = \langle Q, q_0, \Delta, q_f \rangle$ be a 2-counter machine where $Q = \{q_0, q_1, q_f\}$ and

$$\Delta = \{(q_0, \emptyset, I, I, q_1), (q_1, \emptyset, I, D, q_0), (q_1, \emptyset, I, D, q_f)\}$$

Then

$$(q_0, 0, 0)(q_1, 1, 1)(q_0, 2, 0)(q_1, 3, 1)(q_f, 4, 0)$$

is a halting computation using the transitions

$$(q_0, \emptyset, I, I, q_1), (q_1, \emptyset, I, D, q_0), (q_0, \emptyset, I, I, q_1), (q_1, \emptyset, I, D, q_f).$$

Observe that whenever $\mathcal{M}$ halts the first counter has an even value and the second counter is zero. $\star$

Remark In 1961 Marvin L. Minsky introduced 2-counter machines as *two-tape non-writing Turing Machines* [12]: each counter is represented by a semi-infinite tape with a distinct symbol at the beginning of the tape. Writing on the tapes is disallowed altogether. Incrementing or decrementing a counter then corresponds to the read-head being moved right or left respectively. Detecting that a counter is zero corresponds to checking whether the read-head is pointing to the distinct symbol at the beginning of the tape or not. The idea of interpreting the tapes as counters is a more intuitive approach, though, and has become the standard. $\star$

Lemma 3.5 (Minsky [12]) *2-counter machines are Turing-complete.* $\square$

In the next chapter we will reduce the Halting Problem for 2-counter machines to the Universality Problem for timed automata. Undecidability then follows directly from Corollary 3.6.

Corollary 3.6 *The Halting Problem for 2-counter machines is undecidable.* $\square$

3.2 Encoding Computations

We next encode each halting computation from a 2-counter machine into a timed word — or rather a set of timed words. This encoding is strongly inspired by Alur's and Dill's encoding for the undecidability proof in [5] and will play an important role when proving Theorem 4.1 in the next chapter.

Definition 3.7 A *time-encoding* $\psi(s, c, d)$ of a configuration (s, c, d) of $\mathcal{M}$ is a timed word (σ, τ) over the alphabet $\Sigma = Q\dot{\cup}\{a, Z\}$ where $\sigma = sa^cZa^d$ and

- the event s occurs at an integer time $\tau_1 \in \mathbb{N}$,
- the event Z occurs at time $\tau_{c+2} \in (\tau_1, \tau_1 + 1)$,
- the events a^c occur in the interval (τ_1, τ_{c+2}),
- the events a^d occur in the interval $(\tau_{c+2}, \tau_1 + 1)$ and
- no events are simultaneous, i.e. τ is a strongly monotonic time sequence.

$\star$

Definition 3.8 Let $\mathcal{M} = \langle Q, q_0, \Delta, q_f \rangle$ be a 2-counter machine. A *time-encoding* of a halting computation

$$(s_1, c_1, d_1), (s_2, c_2, d_2), \ldots, (s_{l-1}, c_{l-1}, d_{l-1}), (s_l, c_l, d_l)$$

of $\mathcal{M}$ is a timed word (σ, τ) over the alphabet $\Sigma = Q \dot{\cup} \{a, Z\}$ where

$$\sigma = \alpha\psi(s_1, c_1, d_1)\psi(s_2, c_2, d_2) \ldots \psi(s_{l-1}, c_{l-1}, d_{l-1})s_l\beta$$

such that

$$\forall 1 \leq i \leq |\alpha| : (\alpha_i, \tau_i) \in \Sigma \times [0, 1),$$

$\beta \in \Sigma^*$ and

- $\forall 1 \leq i \leq l : s_i$ occurs at time $i \in \mathbb{N}$,
- $\forall 1 \leq i \leq l - 2 : \tau_{i+1}(Z) - \tau_i(Z) = 1$, where $\tau_i(Z)$ denotes the time at which the i^{th} occurrence of Z outside α occurs; this coincides with the Z that occurs in the interval $(i, i + 1)$;

- $\forall 1 \leq i \leq l-1$
  - if $c_{i+1} = c_i + 1$ then for every a at time $t \in (i+1, \tau_{i+1}(Z))$ except the last one, there is an a at time $t-1$;
  - if $c_{i+1} = c_i - 1$ then for every a at time $t \in (i, \tau_i(Z))$ except the last one, there is an a at time $t+1$;
  - if $d_{i+1} = d_i + 1$ then for every a at time $t \in (\tau_{i+1}(Z), i+2)$ except the last one, there is an a at time $t-1$;
  - if $d_{i+1} = d_i - 1$ then for every a at time $t \in (\tau_i(Z), i+1)$ except the last one, there is an a at time $t+1$;

⋆

Remark It is essential that for 2-counter machines no transitions start in the final state q_f. Otherwise one encoding could represent multiple computations. ⋆

Example 3.9 Let $\mathcal{M} = \langle Q, q_0, \Delta, q_f \rangle$ be a 2-counter machine where $Q = \{q_0, q_1, q_f\}$ and

$$\Delta = \{(q_0, \emptyset, I, I, q_1), (q_1, \emptyset, I, D, q_0), (q_1, \emptyset, I, D, q_f)\}$$

Then

$$(q_0, 0, 0)(q_1, 1, 1)(q_0, 2, 0)(q_1, 3, 1)(q_f, 4, 0)$$

is a halting computation, which can be encoded as:

$$(q_0, 1)(Z, 1.5) \quad (q_1, 2)(a, 2.1)(Z, 2.5)(a, 2.7) \quad (q_0, 3)(a, 3.1)(a, 3.2)(Z, 3.5)$$
$$(q_1, 4)(a, 4.1)(a, 4.2)(a, 4.21)(Z, 4.5)(a, 4.6) \quad (q_f, 5)$$

whereas

$$(q_0,1)(Z,1.5)\quad (q_1,2)(\mathbf{a},\mathbf{2.1})(Z,2.5)(a,2.7)\quad (q_0,3)(\mathbf{a},\mathbf{3.05})(\mathbf{a},\mathbf{3.1})(Z,3.5)$$
$$(q_1,4)(a,4.05)(a,4.1)(a,4.21)(Z,4.5)(a,4.6)\quad (q_f,5)$$

is not: the event $(a,3.05)$ does not have a corresponding predecessor $(a,2.05)$ and the event $(a,3.1)$ has a predecessor $(a,2.1)$ although it should not. $\star$

Chapter 4

Universality is Undecidable

We now give a new proof of the undecidability of universality using the same proof idea that Alur and Dill presented in [5]. We essentially reduce the Halting Problem for 2-counter machines to the Universality Problem for timed automata. Note that we use specific timed automata, which we will call *flat timed automata* in Chapter 5, thus leading us to our main result.

Theorem 4.1 *Over both strongly and weakly monotonic time the Universality Problem for timed automata*

- *with at most two clocks and*
- *clock comparisons to the constants 0 and 1 only*

is undecidable.

Proof. Given a 2-counter machine $\mathcal{M} = \langle Q, q_0, \Delta, q_f \rangle$ we can construct a timed automaton $\mathcal{A} = \langle S, \Sigma, C, I, \Gamma, F \rangle$ such that it is universal iff $\mathcal{M}$ does not halt. Such an automaton $\mathcal{A}$ will accept any invalid encoding of a halting computation of $\mathcal{M}$. In particular, if no halting computation of $\mathcal{M}$ exists every encoding is invalid and hence every timed word is accepted.

The alphabet of $\mathcal{A}$ contains one symbol for each state of $\mathcal{M}$ plus two distinct symbols a and Z, i.e. $\Sigma = Q\dot{\cup}\{a, Z\}$. We further define

$$\Sigma' := \Sigma\backslash\{q_f\} \quad \text{and} \quad Q' := Q\backslash\{q_f\}.$$

The states from $\mathcal{A}$ will not be labelled and all state events correspond to a state of $\mathcal{M}$.

We now describe $\mathcal{A}$ by giving several small timed automata. Each of them takes care of a different violation on the encoding. Taking the union of all these automata will result in an automaton that accepts any word that is not a valid encoding of a halting computation of $\mathcal{M}$. Note that we will use three clocks abs, x, y, but two are sufficient: in any instance where we use the clock abs, we could also use the clock x. Due to restrictions on flat timed automata the clock abs is also used in this proof, though.

By Definition 3.8 a time-encoding of a halting computation of $\mathcal{M}$ can only be violated as follows:

1. there were two simultaneous events
2. at time 1 there is no q_0
3. a state event was misplaced or missing
4. a Z was misplaced or missing
5. we did not start with empty counters

6. an a was not matched independent of whether the counter was incremented or decremented
7. an a was illegally inserted
8. a counter was neither incremented nor decremented correctly
    (a) when a counter is not 0
    (b) when a counter is 0
    (c) in the base case (when leaving the first state)
9. a nonexistent transition was taken
    (a) increment both counters, both counters not 0
    (b) increment both counters, one counter 0
    (c) increment both counters, both counters 0
    (d) increment one counter, decrement the other, both counters not 0
    (e) increment one counter, decrement the other, counter to be incremented 0
    (f) decrement both counters
    (g) base case (when leaving the first state)

In the table below we give an overview of where which transition case is handled. We distinguish between the first counter c and the second counter d being $= 0$ or > 0 and whether each counter is incremented or decremented (a_c and a_d). Note that some transitions do not have to be handled, because a counter can only be decremented if it is > 0.

c	d	a_c	a_d		c	d	a_c	a_d		c	d	a_c	a_d		c	d	a_c	a_d	
$=$	$=$	I	I	c)	$=$	$>$	I	I	b)	$>$	$=$	I	I	b)	$>$	$>$	I	I	a)
$=$	$=$	I	D	↯	$=$	$>$	I	D	e)	$>$	$=$	I	D	↯	$>$	$>$	I	D	d)
$=$	$=$	D	I	↯	$=$	$>$	D	I	↯	$>$	$=$	D	I	e)	$>$	$>$	D	I	d)
$=$	$=$	D	D	↯	$=$	$>$	D	D	↯	$>$	$=$	D	D	↯	$>$	$>$	D	D	f)

All the automata will only have final states. So to avoid accepting an encoding of a halting computation we have to avoid reading q_f unless some violation was already detected. Hence most of the time we disallow q_f to be read. The only cases where we explicitly allow q_f is when we just found a violation or when less than one time unit has passed so far.

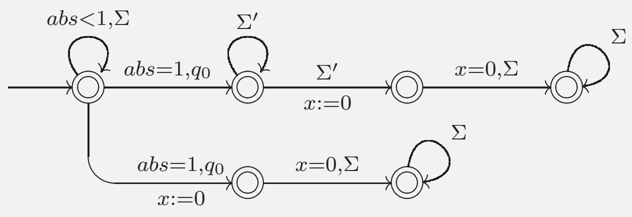

Figure 4.1: simultaneous events, although not allowed

1. First remember that in a valid encoding $\alpha\sigma_1\ldots\sigma_n\beta$ of a halting computation we do not allow any simultaneous events except in α and β. Note that $\sigma_1 = q_0$ and $\sigma_n = q_f$. The automaton in Figure 4.1 accepts any word where at least two simultaneous events occurs: We nondeterministically select an event and reset the clock x. If indeed another event occurs while $x = 0$ then we have found two simultaneous events. The lower branch handles the case where there are at least two simultaneous event at time 1, the upper branch all the other instances.

   Observe that over strongly monotonic time this automaton is not necessary, because simultaneous events are impossible altogether.

2. The automaton in Figure 4.2 accepts any encoding for which at time 1 we do not find q_0, i.e. either at time 1 an event other than q_0 occurs (upper branch) or no event occurs at time 1 at all (lower branch).

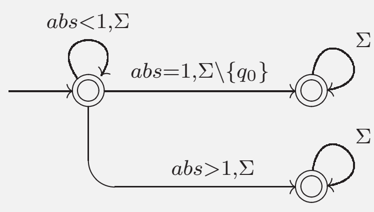

Figure 4.2: no q_0 at time 1

3. The automaton in Figure 4.3 accepts any encoding where there is no state event at an integer time or there is a state event at a non-integer time. In both cases this is only interesting before we see q_f at time $t > 1$. The non-integer time also has to be > 1 for a state event to represent a violation. The automaton in Figure 4.2 accepts any encoding where we do not find q_0 at time 1. Because of this, in the remainder of the proof we will assume that q_0 occurs at time 1.

   Remember that timed automata are nondeterministic. So it is sufficient to find one way of accepting a timed word. In contrast we must not find any way of accepting a timed word if it represents a valid encoding of a halting computation of $\mathcal{M}$.

   We first argue that we can indeed accept any encoding where the property 'state events at integer times' is violated. Observe that if there is at least one such violation we can pick the first one. Assume this violation occurs in the time interval $(l, l+1]$ where $l \in \mathbb{N}$. Then at time l we find a state event. Note that the base case is at time 1 and we already know that q_0 occurs then. The possible violations are:

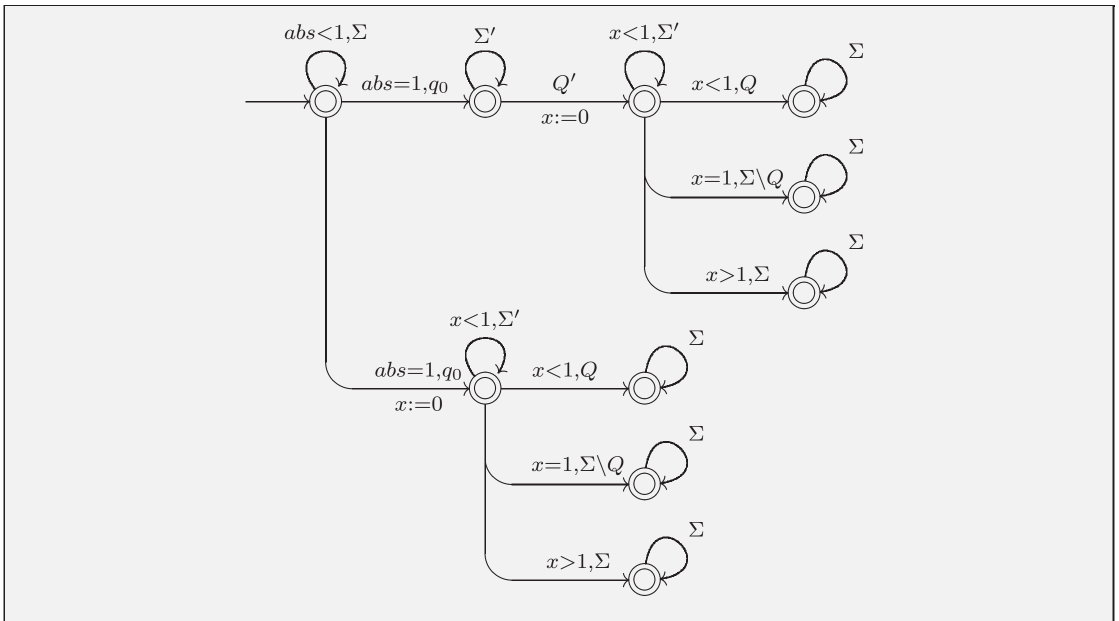

Figure 4.3: not a state event at integer time or state event at non-integer time

- first we might find a state event in $(l, l+1)$, i.e. a state event at a non-integer time;
- second we might find a non-state event at time $l+1$, i.e. no state event at an integer time;
- third maybe no event occurs at time $l+1$ at all, i.e. no state event at an integer time again.

Observe that no event at time $l+1$ can either mean that the last event of the word occurs at time $t < l+1$ or that there is an event at time $t < l+1$ and the next event is at time $t > l+1$.

We now examine how this is realised in the automaton shown in Figure 4.3. We reset the clock x on a nondeterministically selected state event. Assuming that the state event occurs at time l we accept the word according to our explanation above: While $x < 1$ several events can occur. If we actually find a state event while $x < 1$ we may choose to accept because a state event was not allowed. We can also accept if we find a non-state event when $x = 1$. Lastly if no event occurs when $x = 1$ we can also accept.

Note that the three upper branches handle the case where $l > 1$ and the three lower ones where $l = 1$. This differentiation is necessary because in the latter case x has to be reset on q_0 already. Furthermore it is no problem if we nondeterministically choose the wrong state event. If we can accept in at least one way we need not worry about the word anymore. Recall that if a timed word is an invalid encoding due to another violation it is accepted by a different sub-automaton and hence by the automaton $\mathcal{A}$ in whole. Also observe that if the property 'state events at integer times' is not violated then the word cannot be accepted by this sub-automaton. This is due to the fact that we disallow a state event when $x = 1$. If there is no instance where a state event is misplaced we can hence never reach the last state event that allows us to read anything thereafter. As we are not allowed to read q_f when preparing to detect the violation we cannot accidently read a valid encoding, either.

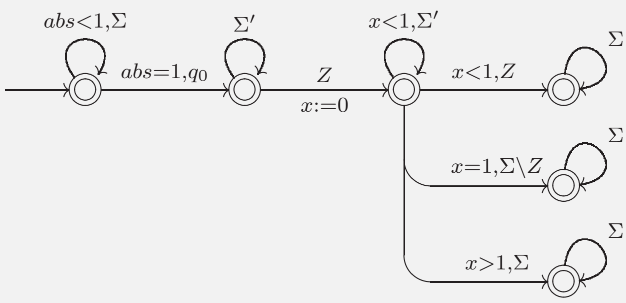

Figure 4.4: not one time unit between each Z

4. Similarly, we want exactly one time unit to pass between each Z. The automaton in Figure 4.4 accepts any encoding where this is not given. The automaton

works with the same principle as the one in Figure 4.3.

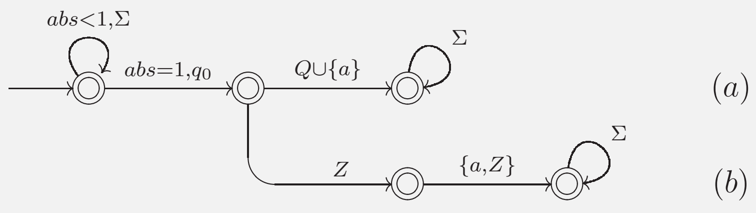

Figure 4.5: did not start with empty counters: (a) Z expected (b) $q_i \in Q$ expected

5. The automaton in Figure 4.5 accepts any encoding that did not start with empty counters. This essentially means if we do not find a $q_0 Z q_i$ starting at time 1 where $q_i \in Q$ then we can accept the word.

6. After having matched the state events and Zs we now handle the counters. We can either increment or decrement a counter. Hence, if c is the value of the first counter, the first $c - 1$ as will definitely have to have a matching successor exactly one time unit later for an encoding to be valid. Figure 4.6 shows an automaton which accepts any encoding where at least one such a is not matched. We reset the clock x on a nondeterministically chosen a. The next event must be an a. This ensures that we did not choose the last a of the counter. To ensure validity when $x = 1$, the next event must be an a. So if there is no event when $x = 1$ we may accept. We may also accept if there is an event when $x = 1$, but it is not an a.

   If we increment the counter, we also have to match the last a of the counter. This will be dealt with later.

7. Another possible violation is that there is an a where we do not expect one to be. If some event occurs at time $t > 1$ and the next event is an a at time

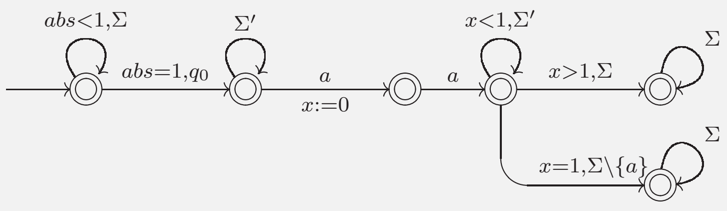

Figure 4.6: not one time unit between matching a, last a ignored due to possible decrement

t', then there is no valid encoding such that there is an event in the interval $(t+1, t'+1)$. This insertion error is detected by the automaton in Figure 4.7. We reset the clock x on a nondeterministically chosen event. The next event has to be an a and on it we reset y. If $x > 1$ and $y < 1$ and an event occurs then we found an event in the interval $(t+1, t'+1)$ and we may hence accept the word.

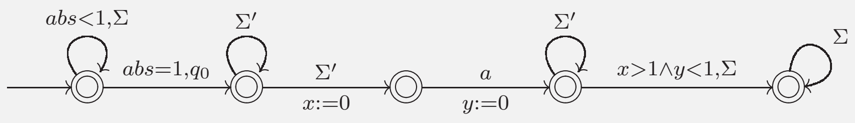

Figure 4.7: insertion error: $\_a \rightarrow \_?a$

8. We next check whether the counters were always incremented or decremented correctly independent of whether a corresponding transition exists or not.

   (a) In Figure 4.8 we give an automaton that accepts any word where the first counter was neither decremented nor incremented correctly. In the upper branch we cannot have decremented because there is an a when $x \geq 1$. But we cannot have incremented correctly either, because then there would have had to be an a when $x = 1$; remember we have to match the last a because this was not done by the automaton in Figure 4.6. In the middle branch we find an a when $x = 1$, so we cannot have decremented. But

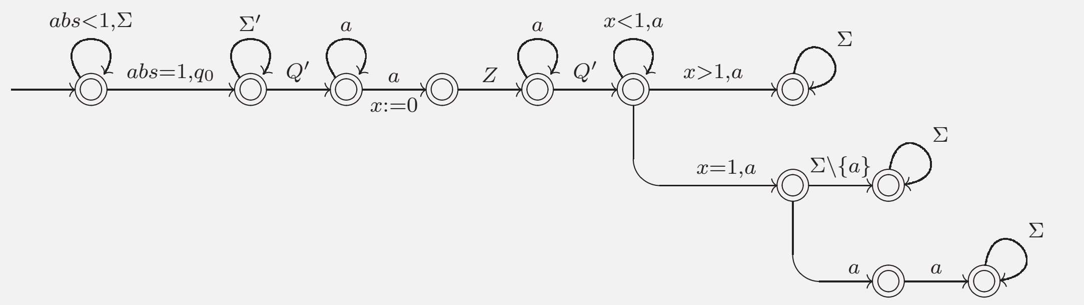

Figure 4.8: first counter > 0 was neither incremented nor decremented, similar for second counter

we cannot have incremented either because no a follows after that. In the lower branch an a follows, so the increment is almost correct. But another a follows, meaning we incremented by 2 instead of only 1.

We can similarly accept any word where the second counter was neither incremented nor decremented correctly.

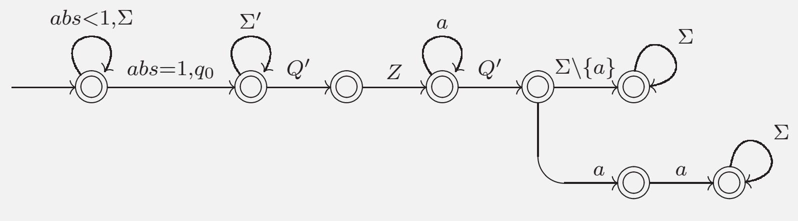

Figure 4.9: first counter $= 0$ was not incremented, similar for second counter

(b) In the previous case we assumed that the first counter was > 0. If the first counter is 0 then the counter must be incremented. Figure 4.9 shows an automaton that accepts any word where the first counter is not incremented in this case. This can similarly be done for the second counter.

(c) The automaton in Figure 4.10 accepts any word where the first or second counter was not incremented in the beginning. Note that in the beginning both counters are empty, so a decrement is not permitted.

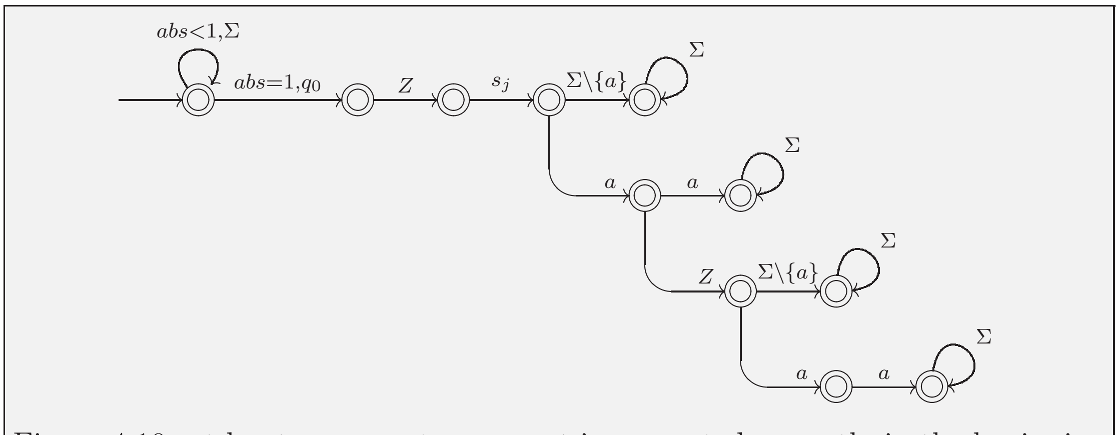

Figure 4.10: at least one counter was not incremented correctly in the beginning

9. After having accepted any words where a counter was not incremented or decremented correctly we concentrate on the transitions. If a nonexistent transition is performed then this represents a violation. In the next few figures we give automata that accept a word where part of the encoding represents a valid realisation of the transition (s_i, r, a_c, a_d, s_j). The chief attraction is that this transition is not included in $\mathcal{M}$. In some cases various restrictions r are possible and we then have to ensure that no transition with any possible r is in Δ.

   (a) The automaton in Figure 4.11 accepts any word that uses the nonexistent transition $(s_i, \emptyset, I, I, s_j)$. Notice that we assume that both the first counter and the second counter are not 0, so the only possible restriction is $\emptyset$.

   (b) The automaton in Figure 4.12 accepts any word that uses the nonexistent transition (s_i, r, I, I, s_j) where r is either $\emptyset$ or $\{c = 0\}$. The automaton only accepts if the first counter is 0, so two restrictions are possible.

   We can similarly accept any word that uses the nonexistent transition (s_i, r, I, I, s_j) where r is either $\emptyset$ or $\{d = 0\}$ by modifying it such that

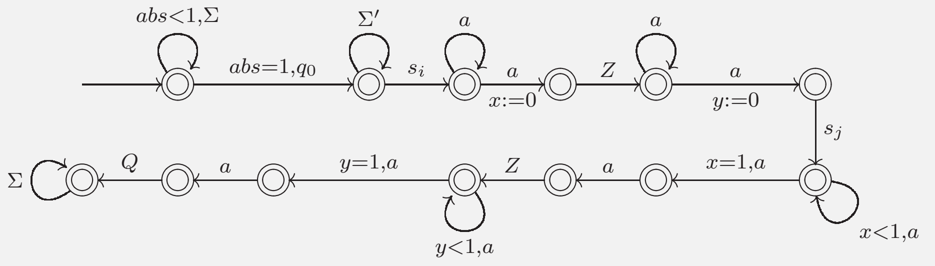

Figure 4.11: if $(s_i, \emptyset, I, I, s_j) \notin \Delta$ where $s_i \in Q'$ and $s_j \in Q$

the first counter is > 0 and the second counter is 0 and increments are performed.

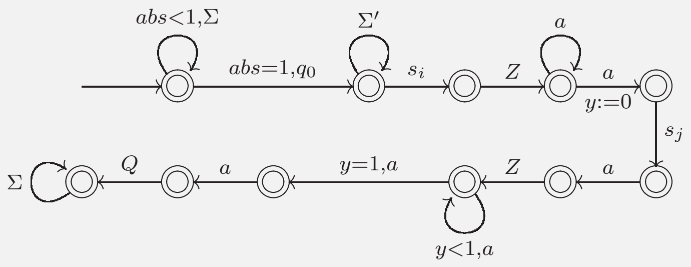

Figure 4.12: if $(s_i, \emptyset, I, I, s_j) \notin \Delta$ and $(s_i, \{c = 0\}, I, I, s_j) \notin \Delta$ where $s_i \in Q'$ and $s_j \in Q$

(c) The automaton in Figure 4.13 accepts any word that uses the nonexistent transition (s_i, r, I, I, s_j) where r is an arbitrary restriction. The automaton only accepts if both the first and the second counter are 0, so all restrictions are permitted.

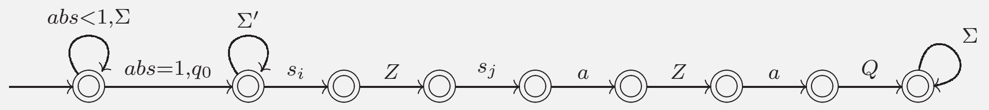

Figure 4.13: if $\forall r \in R : (s_i, r, I, I, s_j) \notin \Delta$ where $s_i \in Q'$ and $s_j \in Q$

(d) The automaton shown in Figure 4.14 accepts any word that uses the nonexistent transition $(s_i, \emptyset, I, D, s_j)$. The second counter is decremented, so it has to be > 0. As the automaton only accepts if the first counter is > 0 the restriction $\emptyset$ is permitted, but no others.

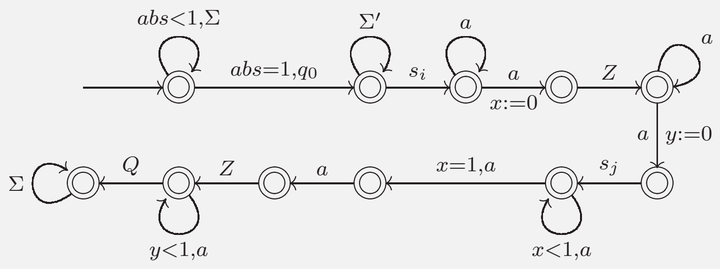

Figure 4.14: if $(s_i, \emptyset, I, D, s_j) \notin \Delta$ where $s_i \in Q'$ and $s_j \in Q$

Similarly we can construct an automaton that accepts any word that uses the nonexistent transition $(s_i, \emptyset, D, I, s_j)$. The automaton will then have to guarantee that the second counter is > 0.

(e) Figure 4.15 shows an automaton that accepts any word that uses the nonexistent transition (s_i, r, I, D, s_j) where r is either $\emptyset$ or $\{c = 0\}$. The automaton only accepts if the first counter is 0, so both restrictions are permitted.

Similarly we can construct an automaton that accepts any word that uses the nonexistent transition (s_i, r, D, I, s_j) where r is either $\emptyset$ or $\{d = 0\}$. The automaton will then have to guarantee that the second counter is 0.

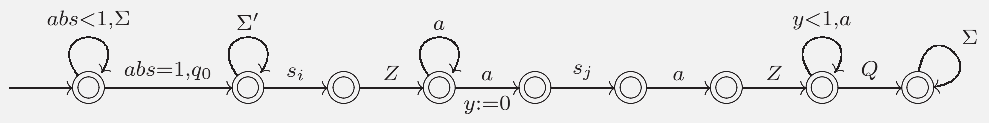

Figure 4.15: if $(s_i, \emptyset, I, D, s_j) \notin \Delta$ and $(s_i, \{c = 0\}, I, D, s_j) \notin \Delta$ where $s_i \in Q'$ and $s_j \in Q$

(f) The automaton in Figure 4.16 accepts any word that uses the nonexistent transition $(s_i, \emptyset, D, D, s_j)$. Observe that both counters have to be > 0

because we want to decrement both of them. Hence the only possible restriction is $\emptyset$.

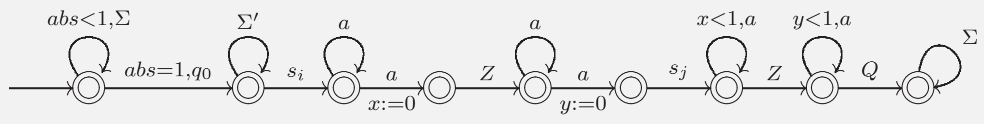

Figure 4.16: if $(s_i, \emptyset, D, D, s_j) \notin \Delta$ where $s_i \in Q'$ and $s_j \in Q$

(g) With the automaton shown in Figure 4.17 we finally check that the first transition taken existed. We already know that both counters are 0 and have to be incremented, so we only need one automaton to capture nonexistent transitions. We further already accepted a word where the two counters are not incremented with the automaton in Figure 4.10, so we can already accept when we find the state event s_j. Also observe that because both counters are empty any restriction is permitted.

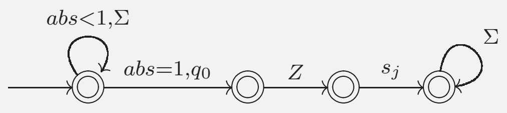

Figure 4.17: if $\forall r \in R : (q_0, r, I, I, s_j) \notin \Delta$ where $s_j \in Q$

In summary we gave several small automata that each detected a different violation on the encoding. Putting them all together we detect all possible violations. So if every timed word is accepted by the constructed timed automaton $\mathcal{A}$, then there exists no valid encoding of a halting computation of $\mathcal{M}$, i.e. there exists no halting computation of $\mathcal{M}$, i.e. $\mathcal{M}$ does not halt. Hence, solving the Universality Problem for timed automata would imply solving the Halting Problem for 2-counter machines.

Observe that we only used the constants 0 and 1 for clock comparisons. We further needed three clocks abs, x and y for all the sub-automata. But as mentioned before we can use the clock x in any instance where we use abs. So two clocks are sufficient. $\square$

Remark We chose to use the clocks abs and x because we will profit from the shape of the sub-automata in Chapter 5. We then do not restrict the number of clocks anymore, but require that the clock abs is never reset. $\star$

Chapter 5

Restricting Resources

In the following we define *flat timed automata* and the *flattening* of a timed language. We then prove an important lemma that describes how a flat timed automaton $\mathcal{A}$ can be converted into a timed automaton with only one state and one symbol such that it accepts the flattening of $\mathcal{L}(\mathcal{A})$. Putting everything together we prove that over weakly monotonic time universality stays undecidable when only allowing clock comparisons to 0 and 1, a single state and a single symbol. We finally give a proof idea on how to extend this result to strongly monotonic time.

5.1 Flat Timed Automata

In the following we will define specific timed automata, which have the property that they can be represented as an collection of linear sub-automata. In Section 5.2 we show how these sub-automata can be converted to timed automata with one state and one symbol only. Of course they cannot accept the same language, because the former allow multiple symbols and the latter only one symbol. We hence define the flattening of a language to establish a connection between the languages.

Definition 5.1 A *flat timed automaton* $\mathcal{A}$ is a 7-tuple $\langle Q, \Sigma, a, q_f, C, I, \Delta\rangle$ such that $\langle Q, \Sigma, C, I, \Delta, F\rangle$ is a timed automaton with the following properties:

- All states are final, i.e. $F = Q$.
- The alphabet includes two special events $a, q_f \in \Sigma$ and we define $\Sigma' := \Sigma \backslash \{q_f\}$.
- A flat timed automaton may not contain any loops apart from the self-loops mentioned below, i.e. we cannot leave a state and later visit it again. Hence we can draw a picture of the automaton such that we either stay stationary or move right, but never left. A simple example is a flat timed automaton that consists of one linear thread as seen in Figure 5.1.

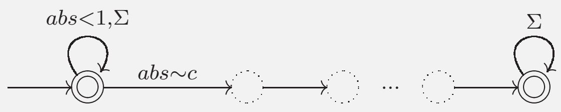

Figure 5.1: Linear-shaped flat automaton

- With inner state we denote a state that can be reached from another state and from which we can go to another state, i.e.

$$q \in Q : \exists s_1, s_2 \in Q \text{ where } s_1 \neq q \neq s_2, \{(s_1, \cdot, \cdot, \cdot, q), (q, \cdot, \cdot, \cdot, s_2)\} \subseteq \Delta.$$

  An *f-state* q of $\mathcal{A}$ is a state $q \in F$ such that it either has no self-loop or the self-loop either allows only the single event a or the whole set of events Σ'. We allow arbitrary restrictions, but no clock resets on the self-loops. All inner states have to be f-states.

- For the edges leading from one state to a distinct state arbitrary restrictions, resets and events are allowed.
- For an edge leading away from an initial state we require that the clock restrictions include '$abs \sim c$' where $abs \in C$ is a clock that is never reset, $\sim \in \{=, >, \geq\}$ and $c \geq 1$. This is needed to initialise all clocks to the value 1.
- Additionally, all initial states $q \in I$ have a fixed self-loop "accept anything within the first time unit", i.e.

$$\forall q \in I, \forall \omega \in \Sigma : (q, \omega, abs < 1, \emptyset, q) \in \Delta.$$

- For any 'end state', i.e. any state without a transition leading to a distinct state, we require a fixed self-loop "accept anything from now on", i.e.

$$\forall q \in Q : [\nexists q' \in Q : (q, \cdot, \cdot, \cdot, q') \in \Delta \rightarrow \forall \omega \in \Sigma : (q, \omega, \texttt{true}, \emptyset, q) \in \Delta].$$

$\star$

Definition 5.2 The *language accepted* by a flat timed automaton $\mathcal{A}$ is the set of all words (σ, τ) such that at least one of the following holds:

- there exists a halting computation for (σ, τ);
- no event occurs at time 1;
- after time 1 at least one full time unit passes between two subsequent events:

$$\exists 1 \leq k < |\sigma| : \quad (\tau_k \geq 1) \wedge (\tau_{k+1} - \tau_k \geq 1);$$

In the following we will call this extension the 'special accept' of $\mathcal{A}$. $\star$

Remark Note that the special accept for flat timed automata does not add to their expressive power. Indeed it can also be seen as an extra restriction: We could also require a clock x that is reset on every event at or after time 1; additionally there would have to be a transition from every inner state to the last state with the only requirement that $(abs > 1) \wedge (x \geq 1)$ is satisfied. We decided not to add the restriction in this form to avoid higher complexity in the description.

We need this special accept when converting to timed automata with one state and one symbol in Section 5.2. When having only one state we use clocks to simulate multiple states. For this we check whether specific clocks are < 1 or ≥ 1. But if one time unit passes without an event occurring necessary information is lost, thus motivating the special accept.

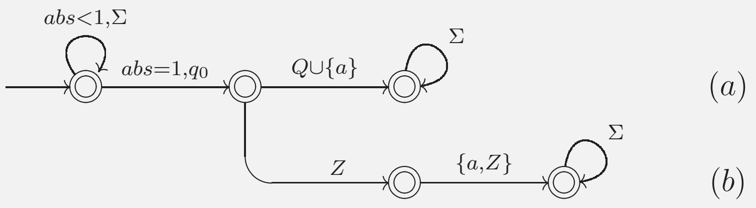

Figure 5.2: flat automaton used in the proof of Theorem 4.1

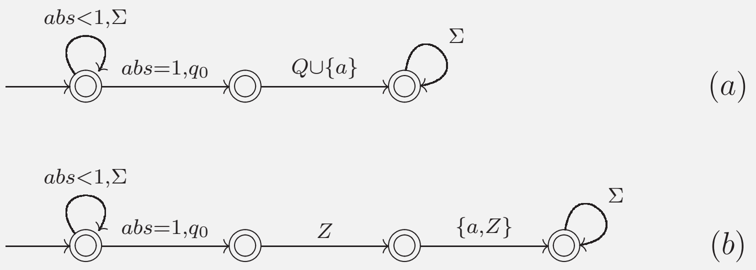

Figure 5.3: purely linear version of the automaton shown in Figure 5.2

Furthermore observe that a flat timed automaton can have a tree-like structure as opposed to the purely linear structure seen in Figure 5.1, the non-existence of loops — apart from self-loops — is then still guaranteed. But without loss of generality we can assume that a flat timed automaton is an collection of purely linear sub-automata: there always exists one such that accepts the same language. See Figure 5.2 and Figure 5.3 for an example.

Also note that the union of two flat timed automata is trivially also a flat timed automaton. This property allows us to specify complex flat timed automata by describing several small flat timed automata. In particular, this was done in the proof of Theorem 4.1. $\star$

Example 5.3 Figure 5.2 shows an automaton used in the proof of Theorem 4.1. Observe that it is indeed a flat timed automaton with the special events a and q_f: the first and last states are as needed and all the inner states are f-states. The only edges leading away from an initial state includes the restriction $abs = 1$. Further, there are no loops apart from the self-loops and all 'end states' accept anything thereafter. $\star$

Remark Note that in no valid encoding (σ, τ) of a halting computation of a 2-counter machine $\mathcal{M}$ can you find k such that

$$\tau_k \geq 1 \quad \wedge \quad \tau_{k+1} - \tau_k \geq 1$$

due to the fact that a state event occurs at every integer time and in between two state events a Z occurs. So (σ, τ) cannot be accepted by the special accept of flat timed automata. Hence we can assume flat timed automata were used in the proof of Theorem 4.1. $\star$

In Section 5.2 we want to convert a flat timed automaton to a timed automaton with one state and one symbol. Clearly the converted automaton cannot accept the same language as the initial flat timed automaton because it uses a different alphabet. Hence we define the flattening of a language in the following. In particular, we require a specific encoding for the special symbols a and q_f of a flat timed automaton.

Definition 5.4 Let $\mathcal{A} = \langle Q, \Sigma, a, q_f, C, I, \Delta \rangle$ be a flat timed automaton. Let

$$\eta : \Sigma \to \{1, \ldots, |\Sigma|\}$$

be an enumeration of Σ such that $\eta(a) = 1$ and $\eta(q_f) = |\Sigma|$. Let S be an arbitrary symbol. Define

$$\Upsilon(\omega, \tau) := \{(S, \tau)^{\eta(\omega)}\}$$

for $(\omega, \tau) \in \Sigma \times \mathbb{R}$ where $\omega \neq q_f$. For $\omega = q_f$ we define $\Upsilon(\omega, \tau)$ to be the set

$$\Upsilon(q_f, \tau) := \{(S, \tau)^{\eta(\omega)}(S, \tau)^k : k \in \mathbb{N}_0\}.$$

Then the *flattening* $f : (\Sigma \times \mathbb{R})^* \to \mathcal{P}((S \times \mathbb{R})^*)$ of Σ is defined by

$$(\sigma, \tau) \mapsto \{\upsilon(\sigma_1, \tau_1)\upsilon(\sigma_2, \tau_2) \ldots | \upsilon(\sigma_i, \tau_i) \in \Upsilon(\sigma_i, \tau_i)\}$$

The *flattening* f of $\mathcal{L}(\mathcal{A})$ is defined by

$$f(\mathcal{L}(\mathcal{A})) := \{f(\sigma, \tau) : (\sigma, \tau) \in \mathcal{L}(\mathcal{A}))\}$$

$\star$

Example 5.5 Let $\Sigma = \{a, b, c, q_f\}$ and η be an enumeration of Σ defined by

$$\eta(a) = 1, \eta(b) = 2, \eta(c) = 3, \eta(q_f) = 4.$$

Then the flattening of

$$(a, 1)(a, 2)(b, 3)(c, 4)(q_f, 5)$$

is the set

$$\{(S, 1)^1(S, 2)^1(S, 3)^2(S, 4)^3(S, 5)^k : k \geq 4\}$$

$\star$

Lemma 5.6 *Let $\mathcal{L}$ be a strongly monotonic language over Σ where $\{a, q_f\} \subseteq \Sigma$. Then*

$$\mathcal{L} \textit{ is strongly timed universal over } \Sigma$$

$$\iff$$

$$f(\mathcal{L}) \textit{ is weakly timed universal over } \{S\}.$$

Proof. '$\Rightarrow$' Assume that $\mathcal{L}$ is strongly timed universal and without loss of generality let $\Sigma = \{a, \omega_2, \ldots, \omega_{k-1}, q_f\}$. It is sufficient to show that for any weakly monotonic word w over $\{S\}$ we can find $(\sigma, \tau) \in \mathcal{L}$ such that $w \in f(\sigma, \tau)$. Given w we can directly determine (σ, τ): For each time t we count the number of simultaneous events. If there is exactly one simultaneous event, this corresponds to an (a, t) in (σ, τ). If there are exactly i simultaneous events where $1 < i < k$ then it corresponds to an (ω_i, t). Finally, if there are at least k simultaneous events, this corresponds to a (q_f, t).

'$\Leftarrow$' Now assume that $f(\mathcal{L})$ is weakly timed universal. It is sufficient to show that for every $(\sigma, \tau) \in \mathcal{L}$ there exists a weakly monotonic word w over $\{S\}$ such that $w \in f(\sigma, \tau)$ and $w \notin f(\sigma', \tau')$ for all other strongly monotonic words (σ', τ'). We can directly determine such a w by encoding each σ_i with exactly $\eta(\sigma_i)$ simultaneous events where $\eta(a) = 1, \eta(\omega_i) = i$ and $\eta(q_f) = k$. $\square$

5.2 Conversion to One State and One Symbol

In the following we will prove that we can convert a flat timed automaton $\mathcal{A}$ to a timed automaton with only one state and one symbol such that it accepts the flattening of $\mathcal{L}(\mathcal{A})$.

Lemma 5.7 *Let $\mathcal{A} = \langle Q, \Sigma, a, q_f, C, I, \Delta\rangle$ be a flat timed automaton. Then we can construct a timed automaton $\mathcal{A}'$ with a single state and a single symbol such that $f(\mathcal{L}(\mathcal{A})) = \mathcal{L}(\mathcal{A}')$.*

Proof. Let $\mathcal{A} = \langle Q, \Sigma, a, q_f, C, I, \Delta\rangle$ be a given flat timed automaton. We first assume that $\mathcal{A}$ consists of only one purely linear thread and generalise afterwards. That is, we assume that $\mathcal{A}$ has exactly one initial state q_0, that every state is reachable from this state — possibly via other states — and that in every state we can reach at most one state that is different from the current one. This essentially means that $\mathcal{A}$ has a shape similar to the one shown in Figure 5.1. Without loss of generality we may assume that for $0 \leq i < |Q| - 1$ the state which we can reach from q_i is called q_{i+1}.

In the following we will construct a timed automaton

$$\mathcal{A}' = \langle \{q\}, \{S\}, C', \{q\}, \Delta', \{q\}\rangle$$

such that it accepts the flattening of $\mathcal{L}(\mathcal{A})$. Having only one symbol S we will have to encode the alphabet Σ of $\mathcal{A}$. For this we will use the flattening of Σ as defined in Definition 5.4.

We first explain which clocks we use in $\mathcal{A}'$, then introduce the invariant "acceptance" and finally explicitly give the transitions for $\mathcal{A}'$ depending on the transition relation Δ of $\mathcal{A}$. Note that for every transition (q, S, ϕ, r_C, q) of $\mathcal{A}'$ we only give the restrictions ϕ and resets r_C; the symbol and the source and target state can be omitted because there is no choice.

The automaton $\mathcal{A}'$ has the set of clocks

$$\begin{aligned} C' = \; & \{abs, event, accept\} \,\dot\cup\, C\backslash\{abs\} \,\dot\cup\, \{B_i : 1 \leq i < |Q|\} \\ & \dot\cup\, \{M_{i,j} : 0 \leq i \leq |Q|, 1 \leq j \leq |\Sigma|\} \,\dot\cup\, \{sim_j : 1 \leq j \leq |\Sigma|\}. \end{aligned}$$

The clock *abs* is never reset, it helps us keep track of how much time has passed already. The clock *event* will be reset on every event where $abs \geq 1$, it is used for checking when the last event occurred. The clock *accept* is reset if a word can be accepted with the special accept of flat timed automata. The B_i will help us simulate states: if $B_i < 1$ and $B_{i+1} \geq 1$, then we are simulating the state q_i of $\mathcal{A}$ where $0 < i < |Q| - 1$. For the last state $q_{|Q|-1}$ we only require $B_{|Q|-1} < 1$. The clocks $M_{i,j}$ help us express which events are allowed when going from q_i to q_{i+1} and the clocks sim_j handle the encodings for self-loops.

Recall that flat timed automata have a special accept: if at least one time unit passed after time 1 without an event occurring, then the word is accepted. For converting this special accept we introduce an invariant which we call "acceptance". Once the invariant is established it can be maintained. We accomplish this with the transitions shown in Transition 5.1.

Observe that if at least one time unit passes between two events, then $event \geq 1$ and we trigger acceptance. Then two settings are possible: either another event occurs before one time unit and $accept < 1$; or at least one time unit passes before

$restriction$	$resets$
$abs > 1 \wedge (event \geq 1 \vee accept < 1)$	$accept, event$

Transition 5.1: Invariant “acceptance”

the next event occurs and $event \geq 1$. Notice that as soon as acceptance was triggered $accept = event$ will hold. Hence, whenever $accept \geq 1$ it is guaranteed that $event \geq 1$, too. Hence, once acceptance is established the word can be accepted. As we only need one halting computation to accept a word, we may in the following assume that at all times after 1 before one full time unit passes an event occurs. In particular, if B_i was reset we can assume that $B_i < 1$ on the next event.

We next specify further transitions for $\mathcal{A}'$ depending on which transitions are included in $\mathcal{A}$. In general, for each transition of $\mathcal{A}$ we need a set of transitions in $\mathcal{A}'$. These usually have a common prefix

$$cond := \phi \wedge (abs \geq 1) \wedge (B_i < 1) \wedge (B_{i+1} \geq 1)$$

where ϕ are the restrictions of the transition in $\mathcal{A}$, $abs \geq 1$ guarantees that at least one time unit has already passed and $(B_i < 1) \wedge B_{i+1}$ ensures that we are simulating the state q_i of $\mathcal{A}$.

Further transitions that are included in Δ' are now given dependent on the types of transitions found in Δ.

1. Anything may happen before time 1

   In $\mathcal{A}$ this was represented by transitions

$$\forall \omega \in \Sigma : (q_0, \omega, abs < 1, \emptyset, q_0) \in \Delta.$$

In $\mathcal{A}'$ this is represented by the transition shown in Transition 5.2.

restrictions	*resets*
$abs < 1$	–

Transition 5.2: Accept anything before time 1

2. Transitions to a different state with symbol 'a'

   These are represented by transition of the shape $(q_i, a, \phi, r_C, q_{i+1}) \in \Delta$. The corresponding transition for $\mathcal{A}'$ where $i \neq 0$ is given in Transition 5.3. The

$$cond := \phi \wedge (abs \geq 1) \wedge (B_i < 1) \wedge (B_{i+1} \geq 1)$$

restriction	*resets*
$cond \wedge (event > 0)$	$\forall x \in r_C : x, B_1, \ldots, B_{i+1}, event$

Transition 5.3: Corresponding to $(q_i, a, \phi, r_C, q_{i+1}) \in \Delta$

   restriction $(B_i < 1) \wedge (B_{i+1} \geq 1)$ expresses that we are currently in the state q_i. The restriction $event > 0$ ensures that no event has happened at that time yet. This is important because a was encoded by a single S.

3. Transitions to a different state with symbol 'q_f'

   These are represented by transition of the shape $(q_i, q_f, \phi, r_C, q_{i+1}) \in \Delta$. The corresponding transitions for $\mathcal{A}'$ where $i \neq 0$ are given in Transition 5.4 where q_f is encoded by $S^k S^*$. Again the restrictions $(B_i < 1) \wedge (B_{i+1} \geq 1)$ express that we are currently in the state q_i. The restriction $event > 0$ on the first transition ensures that no event has happened at that time yet. We then count how many simultaneous events occur. If at least k simultaneous events occurred, we are allowed to reset the clock B_{i+1}, which signals a state change to q_{i+1}. Notice

$$cond := \phi \wedge (abs \geq 1) \wedge (B_i < 1) \wedge (B_{i+1} \geq 1)$$

restriction	*resets*
$cond \wedge (event > 0)$	$\forall x \in r_C : x, B_1, \ldots, B_i, event, M_{i,1}$
$cond \wedge (M_{i,1} = 0) \wedge (M_{i,2} > 0)$	$event, M_{i,2}$
$cond \wedge (M_{i,2} = 0) \wedge (M_{i,3} > 0)$	$event, M_{i,3}$
$\vdots$	$\vdots$
$cond \wedge (M_{i,k-1} = 0) \wedge (M_{i,k} > 0)$	$event, M_{i,k}$
$cond \wedge (M_{i,k} = 0)$	$event$
$cond \wedge (M_{i,k-1} = 0 \vee M_{i,k} = 0)$	$event, B_{i+1}$

Transition 5.4: Corresponding to $(q_i, q_f, \phi, r_C, q_{i+1}) \in \Delta$ where $\eta(q_f) = k$

that we have to non-deterministically select the last S that encodes q_f for the reset of B_{i+1}, because the next transition will require $event > 0$.

4. Transitions to a different state with any other symbol

   These are represented by transition of the shape $(q_i, \omega, \phi, r_C, q_{i+1}) \in \Delta$ where $\omega \in \Sigma \backslash \{a, q_f\}$. The corresponding transitions for $\mathcal{A}'$ where $i \neq 0$ are given in Transition 5.5 where ω is encoded by S^n. This is an easier version of the case

$$cond := \phi \wedge (abs \geq 1) \wedge (B_i < 1) \wedge (B_{i+1} \geq 1)$$

restriction	*resets*
$cond \wedge (event > 0)$	$\forall x \in r_C : x, B_1, \ldots, B_i, event, M_{i,1}$
$cond \wedge (M_{i,1} = 0) \wedge (M_{i,2} > 0)$	$event, M_{i,2}$
$cond \wedge (M_{i,2} = 0) \wedge (M_{i,3} > 0)$	$event, M_{i,3}$
$\vdots$	$\vdots$
$cond \wedge (M_{i,n-2} = 0) \wedge (M_{i,n-1} > 0)$	$event, M_{i,n-1}$
$cond \wedge (M_{i,n-1} = 0) \wedge (M_{i,n} > 0)$	$event, B_{i+1}$

Transition 5.5: Corresponding to $(q_i, \omega, \phi, r_C, q_{i+1}) \in \Delta$ where $\eta(\omega) = n$

   for q_f, because we know how many 'S'-symbols encode ω.

5. Transitions leading away from the first state

These are represented by transition of the shape $(q_0, \omega, \phi, r_C, q_1) \in \Delta$. They are very similar to the ones presented in Transition 5.3, 5.4 and 5.5 where $i = 0$, but the common prefix for the restrictions is

$$cond := \phi \wedge (abs \geq 1) \wedge (B_1 \geq 1).$$

This signalises that we have not reset any state clock yet, i.e. that we are still in the first state. We cannot use the same mechanism as in the instances above, because we need the first time unit to initialise all clocks.

6. Self-loops with 'a'

   These self-loops are represented by transition of the shape $(q_i, a, \phi, \emptyset, q_i) \in \Delta$. Remember that we did not allow any resets on the self-loops. The corresponding transitions for $\mathcal{A}'$ are given in Transition 5.6. Note that we require the restriction $(M_{i,1} \geq 1)$ to ensure that we did not reset $M_{i,1}$ yet. Else we non-deterministically decided that the transition to the next state is going to be taken and a self-loop is not allowed anymore.

$$cond := \phi \wedge (abs \geq 1) \wedge (B_i < 1) \wedge (B_{i+1} \geq 1)$$

restriction	*resets*
$cond \wedge (event > 0) \wedge (M_{i,1} \geq 1)$	$B_1, \ldots, B_i, event$

Transition 5.6: Corresponding to self-loops with a

7. Self-loops with Σ'

   These self-loops are represented by transitions of the shape $\forall \omega \in \Sigma \backslash \{q_f\} : (q_i, \omega, \phi, \emptyset, q_i) \in \Delta$. The corresponding transitions for $\mathcal{A}'$ are given in Transition 5.7 where q_f is encoded by $S^k S^*$. Here we basically count the number of

$$cond := \phi \land (abs \geq 1) \land (B_i < 1) \land (B_{i+1} \geq 1)$$

restriction	*resets*
$cond \land (event > 0) \land (M_{i,1} \geq 1)$	$B_1, \ldots, B_i, event, sim_1$
$cond \land (sim_1 = 0) \land (sim_2 > 0)$	$event, sim_2$
$cond \land (sim_2 = 0) \land (sim_3 > 0)$	$event, sim_3$
$\vdots$	$\vdots$
$cond \land (sim_{k-2} = 0) \land (sim_{k-1} > 0)$	$event, sim_{k-1}$

Transition 5.7: Corresponding to self-loops with Σ' where $\eta(q_f) = k$

simultaneous events with the clocks sim_j. As before we require that $(M_{i,1} \geq 1)$ to ensure that we did not reset $M_{i,1}$ yet. Note that we never allow an event to occur when $sim_{k-1} = 0$. This prevents that q_f occurs in a self-loop.

8. Self-loops with Σ

   These transition can only occur in end-states, i.e. those that cannot be left anymore. In our instance this is $q_{|Q|-1}$. The self-loop is represented by the transitions

   $$\forall \omega \in \Sigma : (q_{|Q|-1}, \omega, \texttt{true}, \emptyset, q_{|Q|-1}) \in \Delta.$$

   The corresponding transition for $\mathcal{A}'$ is given in Transition 5.8.

restriction	*resets*
$(abs \geq 1) \land (B_{|Q|-1} < 1)$	$B_1, \ldots, B_{|Q|-1}, event$

Transition 5.8: Corresponding to self-loops with a

After having explained how one purely linear thread is converted it is easy to generalise this to all flat timed automata. Recall that for every flat timed automaton there exists a flat timed automaton such that it accepts the same language and it is an assemblage of purely linear threads. So without loss of generality we can assume that $\mathcal{A}$ is an assemblage of purely linear threads. Then we can convert each thread separately and take the union of the resulting automata. Note that we still only have one state and one symbol. However, we have to guarantee that each thread has its own set of clocks. More precisely, it is sufficient if the 'state clocks' B_i are distinct for each thread. This can be accomplished by using $B_{i,j}$ to indicate which thread the clock belongs to. $\square$

Remark When converting flat timed automata to timed automata with one state and one symbol we need all constants that were used in the flat timed automaton for the clock comparisons. Furthermore, we need the constants 0 and 1. In particular, if the given flat timed automaton included comparisons to 0 and 1, then the resulting automaton does not need additional constants. Therefore we just need the constants 0 and 1 when converting the flat timed automata that were used in the proof of Theorem 4.1. $\star$

Example 5.8 To clarify the construction seen in the proof of Lemma 5.7 we give $\mathcal{A}'$ for the flat timed automaton $\mathcal{A} = \langle Q, \{a, b, c, q_f\}, a, q_f, C, I, \Delta\rangle$ as seen in Figure 5.4. We will encode a by S, b by SS, c by SSS and q_f by $SSSSS^*$. All transitions of $\mathcal{A}'$ are listed in Transition 5.9.

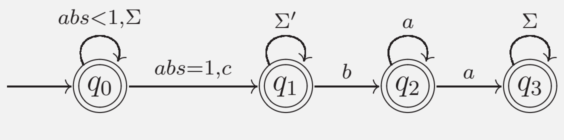

Figure 5.4: Automaton to be transformed

Now let us look at two timed words , the first

$$w_1 = (c, 1)(a, 1.5)(b, 2.0)(a, 2.5)(a, 3.0)(a, 3.5)(b, 4.0)$$

transition		*restriction*	*resets*
	Δ_0	$abs > 1 \wedge (event \geq 1 \vee accept < 1)$	$accept, event$
$abs < 1, \Sigma$	Δ_1	$abs < 1$	$-$
$abs = 1, c$		$abs = 1 \wedge B_1 \geq 1$	
	Δ_2	$-\ event > 0$	$event, M_{0,1}$
	Δ_3	$-\ M_{0,1} = 0 \wedge M_{0,2} > 0$	$event, M_{0,2}$
	Δ_4	$-\ M_{0,2} = 0 \wedge M_{0,3} > 0$	$event, B_1$
Σ'-loop		$abs \geq 1 \wedge B_1 < 1 \wedge B_2 \geq 1 \wedge$	
	Δ_5	$-\ event > 0 \wedge M_{1,1} \geq 1$	$B_1, sim_1, event$
	Δ_6	$-\ sim_1 = 0 \wedge sim_2 > 0$	$sim_2, event$
	Δ_7	$-\ sim_2 = 0 \wedge sim_3 > 0$	$sim_3, event$
b		$abs \geq 1 \wedge B_1 < 1 \wedge B_2 \geq 1 \wedge$	
	Δ_8	$-\ event > 0$	$B_1, event, M_{1,1}$
	Δ_9	$-\ M_{1,1} = 0 \wedge M_{1,2} > 0$	$event, B_2$
a-loop	Δ_{10}	$abs \geq 1 \wedge B_2 < 1 \wedge B_3 \geq 1 \wedge$	
		$event > 0 \wedge M_{2,1} \geq 1$	$B_1, B_2, event$
a	Δ_{11}	$abs \geq 1 \wedge B_2 < 1 \wedge B_3 \geq 1 \wedge event > 0$	$B_1, B_2, B_3, event$
Σ-loop	Δ_{12}	$abs \geq 1 \wedge B_3 < 1$	$B_1, B_2, B_3, event$

Transition 5.9: Transitions for $\mathcal{A}'$

is accepted by $\mathcal{A}$ and the second

$$w_2 = (c, 1)(q_f, 1.5)$$

is not accepted by $\mathcal{A}$.

This translates to

$$f(w_1) = (S, 1)^3(S, 1.5)(S, 2)^2(S, 2.5)(S, 3)(S, 3.5)(S, 4)^2$$

and

$$f(w_2) = (S, 1)^3(S, 1.5)^4(S, 1.5)^*$$

We will now verify that $f(w_1)$ is accepted by $\mathcal{A}'$ and $f(w_2)$ is not accepted by $\mathcal{A}'$. In Transition 5.10 we give the transitions and value of the clocks of a halting computation of $\mathcal{A}'$.

abs	*event*	B_1	B_2	B_3	$M_{0,1}$	$M_{0,2}$	$M_{0,3}$	$M_{1,1}$	$M_{1,2}$	$M_{2,1}$	Δ
1	1	1									Δ_2
1		1			0	1					Δ_3
1		1				0	1				Δ_4
1.5	0.5	0.5	1.5					1.5			Δ_5
2	0.5	0.5	2								Δ_8
2		0	2					0	2		Δ_9
2.5	0.5		0.5	2.5						2.5	Δ_{10}
3	0.5		0.5	3						3	Δ_{10}
3.5	0.5		0.5	3.5							Δ_{11}
4				0.5							Δ_{12}
4				0							Δ_{12}

Transition 5.10: Relevant values of clocks when accepting $f(w_1)$

We now try to accept $f(w_2)$ and can get as far as seen in Transition 5.11. But

abs	*event*	B_1	B_2	B_3	$M_{0,1}$	$M_{0,2}$	$M_{0,3}$	$M_{1,1}$	sim_1	sim_2	sim_3	Δ
1	1	1										Δ_2
1		1			0	1						Δ_3
1		1				0	1					Δ_4
1.5	0.5	0.5	1.5					1.5				Δ_5
1.5		0	1.5						0	1.5		Δ_6
1.5		0	1.5							0	1.5	Δ_7
1.5	0	0	1.5	1.5				1.5	0	0	0	$\lightning$

Transition 5.11: Values of clocks for $f(w_2)$

after taking transition Δ_7, we cannot go on: $event = 0$ and $accept = 1.5$, so Δ_0 is not allowed. $abs = 1.5$, so Δ_1 is not allowed. Δ_2, Δ_3 and Δ_4 are not allowed, because

$B_1 = 0$. Δ_5, Δ_6 and Δ_7 are not allowed, because $event = 0$, $sim_2 = 0$ and $sim_3 = 0$ respectively. Similarly Δ_8 and Δ_9 are not allowed. Δ_{10} and Δ_{11} are not allowed, because $B_2 = 1.5$. Finally, Δ_{12} is not allowed, because $B_3 = 1.5$.

Note that by taking other transitions than seen in Transition 5.11 we we cannot go on at an even earlier point. Hence the timed word $f(w_2)$ is not accepted. $\star$

Corollary 5.9 *A flat timed automaton $\mathcal{A}$ is strongly timed universal iff its converted automaton $\mathcal{A}'$ with one state and one symbol is weakly timed universal.*

Proof. This follows from Lemma 5.6 and Lemma 5.7. $\square$

5.3 Main Result

Theorem 5.10 *Over weakly monotonic time universality is undecidable for timed automata with a single state and a single symbol using only comparisons to the constants 0 and 1.*

Proof. In the proof of Theorem 4.1 we only used flat timed automata. Hence this theorem directly follows from Theorem 4.1 and Corollary 5.9. $\square$

5.4 Strongly Monotonic Time

We believe a slightly weaker version of our result holds over both weakly and strongly monotonic time.

Conjecture 5.11 *Universality is undecidable for timed automata with a single state and a single symbol for both weakly and strongly monotonic time.*

Proof idea. Observe that our time-encoding of 2-counter machines did not need any simultaneous events. Only when introducing the flattening did these begin to play a leading role.

Instead of encoding the state symbols with simultaneous events we can encode them with events that are exactly one time unit apart. The end of a state symbol encoding is then not signalled by time passing, but by either finding a counter symbol, which occurs at a non-integer time, or the passage of $|Q| + 2$ time units. The counter values are still encoded by single events as before.

Now remember we had to match the 'a's for the counter values. How can this still be done if we do not exactly know at what time to look? First of all, we require that any event for encoding state symbols occurred at an integer time. Any event for the value of one of the counters will occur at a non-integer time. So we can also reset a specific clock y if we find an event at a non-integer time. Additionally, we only have finitely many states and hence an encoding of a state symbol will only need a limited amount of time. So when checking for matching 'a's we need only to check that our clock y has an integer value within a specific range, e.g. with a constraint like

$$(y = 6) \vee (y = 7) \vee (y = 8).$$

Now we also have to decide whether an a belonged to the first or the second counter. For this we reset a clock x on the last S encoding a state. For the first counter we then change the constraint to

$$(5 < x < 6 \wedge y = 6) \vee (6 < x < 7 \wedge y = 7) \vee (7 < x < 8 \wedge y = 8).$$

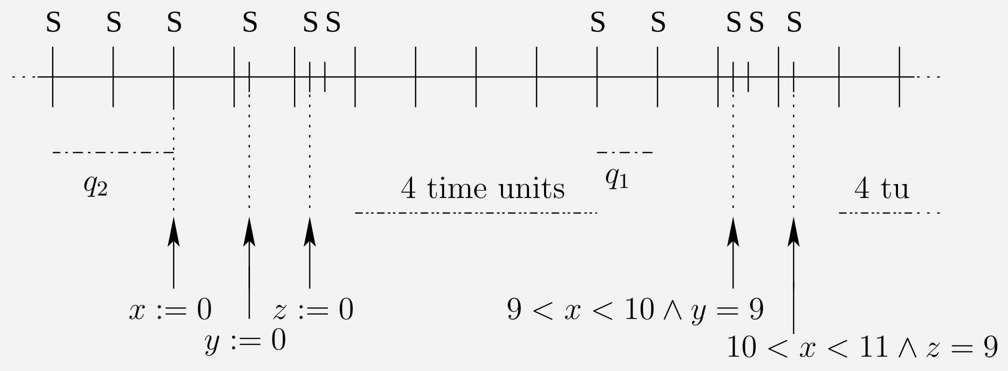

Figure 5.5: Matching counter events

Similarly, for the second counter this constraint would have the shape

$$(6 < x < 7 \wedge y = 6) \vee (7 < x < 8 \wedge y = 7) \vee (8 < x < 9 \wedge y = 8).$$

For a visualisation see Figure 5.5.

Note that we let $|Q|$ time units pass between two configurations. This assures that we cannot skip a whole configuration when matching the counter events.

This basic idea does not need simultaneous events. Hence the proof works for both weakly and strongly monotonic time: the only difference is that for strongly monotonic time we immediately accept if some simultaneous event occurs. Observe though, that the number of constants needed is linear in $|Q|$ opposed to two as seen before. $\square$

Chapter 6

Discussion

We conclude our work by summing up our results, giving a quick glimpse of problems that occurred when working on our proofs and finally raising questions for future work.

6.1 Results

In the previous sections we showed that over weakly monotonic time universality is undecidable for timed automata with at most two clocks and clock comparisons to 0 and 1. Our main result consisted of proving that the problem stays undecidable when restricting to clock comparisons to 0 and 1, one state and one symbol. Hence, we narrowed the frontier between decidability and undecidability considerably.

We believe that for weakly monotonic time resources cannot be restricted further if undecidability of universality is to be maintained. Surely we are optimal in number of states and events and only two constants are used. If we use no more than the constant 0, then it was previously shown that universality is decidable [11]. If we allow no more than one constant $c > 0$ we can no longer detect simultaneous events. The shape of the clock constrains could be restricted. But previous results narrow the possibilities already: in [10] Ouaknine and Worrell showed that for open automata ($<$, $>$ and $\neq$) with arbitrarily many states and symbols universality is decidable. Most definitely limiting the number of clocks is not of interest: if additionally restricting to a finite number of clocks only finitely many timed automata are possible and hence the problem is trivially decidable.

For strongly monotonic time we proposed that universality is undecidable when restricting to one state and one symbol. However, we needed more constants, the number being linearly dependent on the number of states of the given 2-counter machine. We believe this result can be improved, but a different approach might be necessary.

6.2 False starts

A number of findings during the research phase of this project challenged the claim that universality was actually undecidable over weakly monotonic time with our restriction on state and symbol number. We started off by proving that universality was undecidable and then altered the proof to timed automata over weakly monotonic time with only one symbol. This was done in a straightforward fashion by encoding the alphabet using simultaneous events and introducing multiple new states. For timed automata with only one state but several symbols we could introduce extra symbols to help us keep track of what state we were simulating. Upon restricting both symbols and states all this information had to be represented by the clocks. So it is not very surprising that we encountered some problems. One, in particular, fundamentally changed the realization of our project:

For a rather long time we tried to prove our result using channel machines. A channel machine basically is a finite-state machine with a queue: you can write something after the last symbol of the tape or read the first symbol on the tape and remove it. The advantage is that we only have to keep track of one tape instead of two counters and the operations are still very simple. Additionally, channel machines are also Turing-complete. But it turned out that finding a good encoding for the tape alphabet was not so easy.

Remember that for flat timed automata we allowed either an a or all Σ' as events on the self loops in inner states. For the 'only a' case we did not allow any simultaneous events within the self-loop. For the Σ' case we allowed up to k simultaneous events where $S^{k+1}S^*$ encoded the final state. More generally, we could have allowed upwards closed sets of encodings, i.e. up to l simultaneous events for some $l \in \mathbb{N}$.

Unfortunately, we would have needed some self-loops where all tape symbols but one were allowed to detect specific violations on our encoding. This cannot be realised by upwards closed sets in every case. Ultimately, after quite a few attempts, we decided to discard the idea of using channel machines and settled on 2-counter machines. They do not require a tape alphabet and hence the problem was circumvented.

Also notice that our 2-counter machines only have one final state. This does not affect the expressive power of the machines, but it is essential for our proof. We encoded the final state with S^kS^*, i.e. if we found at least k simultaneous S, then we found an encoding for the final state. This is not extendible to multiple final states.

6.3 Future Work

We showed that over weakly monotonic time universality is undecidable for timed automata with clock comparisons to 0 and 1, a single state and a single symbol. As mentioned above we believe that additional restrictions lead to decidability. It has not been proved, though, that restricting the clock constraints further leads to decidability. Although open timed automata are not an option, closed finite automata might prove sufficient for undecidability. Another possibility might be to allow clock comparisons to 1 only. Answers to these questions will lead us to the exact border between decidability and undecidability.

We also gave a proof idea that for strongly monotonic time universality is undecidable with one state and one symbol. This proof should be completed and studied in detail. Furthermore, a valid question is whether universality is maintained when additionally restricting to clock comparisons to 0 and 1 as in the weakly monotonic case. More generally, this research raises the question where the border between decidability and undecidability is for strongly monotonic time.

Bibliography

[1] Rajeev Alur and Thomas A. Henzinger. A Really Temporal Logic. *Journal of the ACM*, 41(1):181–203, 1994.

[2] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-Checking in Dense Real-Time. *Information and Computation*, 104(1):2–34, 1993.

[3] Klaus Havelund, Arne Skou, Kim Larsen, and Kristian Lund. Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pages 2–13, 1997.

[4] Rachel Cardell-Oliver. Conformance Testing of Real-Time Systems with Timed Automata, 1999.

[5] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[6] Thomas A. Henzinger, Vineet Gupta, and Radha Jagadeesan. It's About Time: Real-Time Logics Reviewed. In *Lecture Notes in Computer Science: HART'97*, volume 1201, pages 331–345. Springer-Verlag, 1997.

[7] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Robust Timed Automata. *Theoretical Computer Science*, 211:253–273, 1999.

[8] Dragan Bošnački. Digitization of Timed Automata. In *Proceedings of FMICS'99*, pages 283–302, 1999.

[9] Joël Ouaknine and James Worrell. Revisiting Digitization, Robustness, and Decidability for Timed Automata. In *Logic in Computer Science 2003*, pages 198–207, 2003.

[10] Joël Ouaknine and James Worrell. Universality and Language Inclusion for Open and Closed Timed Automata. In *Lecture Notes in Computer Science: HSCC*, volume 2623, pages 375–388. Springer-Verlag, 2003.

[11] Joël Ouaknine and James Worrell. On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap. In *Logic in Computer Science 2004*, pages 54–63, 2004.

[12] Marvin L. Minsky. Recursive Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines. *Annals of Mathematics*, 74(3):437–455, 1961.