

# Unfoldings for Message Passing Timed Automata

Dirk Fahland

Diplomarbeit



Humboldt-Universität zu Berlin  
Mathematisch-Naturwissenschaftliche Fakultät II  
Institut für Informatik

Betreuer:  
Prof. Dr. W. Reisig  
Prof. P.S. Thiagarajan

Berlin, den 14. Juli 2006



## Zusammenfassung

In dieser Arbeit entwickeln wir auf der Basis von Entfaltungen eine Zustandsraum-Reduktionsmethode für Netzwerke von Zeitautomaten, um der durch nebenläufig aktivierte Aktionen ausgelösten *Zustandsraumexplosion* zu begegnen.

Gängige *Model-Checking-Verfahren* konstruieren zur Verifikation eines Systems dessen sequentiellen Zustandsraum, der exponentiell wächst, wenn das Verfahren auf verteilte Systeme angewandt wird, die nebenläufig aktivierte, unabhängige Aktionen enthalten: Während der Konstruktion des Zustandsraums werden diese Aktionen in jeder beliebigen Reihenfolge geordnet, um ihre Nebenläufigkeit im sequentiellen Modell zu simulieren. Für zeitlose Systeme werden Zustandsraum-Reduktionsmethoden wie *sture Mengen*, die während der Konstruktion redundante Informationen weglassen, oder *Entfaltungen*, die nebenläufige Ereignisse des Systems in einer Halbordnung darstellen, erfolgreich eingesetzt, um das exponentielle Wachstum abzuschwächen.

Die Methoden nutzen ein einfaches, syntaktisches Kriterium um unabhängige Aktionen in System zu erkennen. Dieses Kriterium ist für Netzwerke von Zeitautomaten nicht anwendbar, wie bereits einfache Beispiele zeigen, weshalb verfügbare Techniken für zeitlose Systeme in diesem Fall nicht genutzt werden können. Da bei der Verifikation von Netzwerken von Zeitautomaten jedoch ebenfalls die Zustandsraumexplosion auftritt, besteht die Notwendigkeit einer spezifischen Reduktionsmethode für diese Systeme.

In dieser Arbeit betrachten wir eine spezielle, praktisch relevante Klasse von Netzwerken von Zeitautomaten als ein Modell für diskrete, verteilte, zeitbehaftete Systeme. Wir entwickeln eine neuartige Technik, die eine vollständige, endliche Repräsentation des Zustandsraums eines solchen Systems konstruiert. Diese Repräsentation ist der vollständige, endliche Präfix einer Entfaltung, in der nebenläufige Aktionen halbgeordnet sind. Wir zeigen, dass wir mit dieser Technik in der Lage sind, die Darstellung des Zustandsraums um Größenordnungen zu reduzieren. Uns ist derzeit keine Zustandsraum-Reduktionsmethode bekannt, die vergleichbare Ergebnisse für zeitbehaftete Systeme liefert.



## Abstract

In this thesis we develop a state space reduction technique for *networks of timed automata* based on *unfoldings* to alleviate the *state space explosion problem* due to concurrently enabled actions.

For the purpose of verifying a system, standard *model checking* techniques construct its sequential state space that suffers an exponential growth when applied to distributed systems because of concurrently enabled, independent actions: during the construction of the state space these actions are ordered arbitrarily to simulate concurrency in the sequential model. For untimed systems, state space reduction techniques like *stubborn sets* that omit the construction of redundant information, and unfoldings that represent concurrent events in a partial order have successfully been applied to alleviate the exponential growth.

These techniques apply a simple syntactical criterion to identify independent actions. This criterion is not applicable to networks of timed automata as simple examples show, which renders the existing techniques unapplicable. But networks of timed automata face the state space explosion problem as well which raises the demand for a specific reduction technique for these systems.

In this thesis, we consider a special, but practically relevant class of networks of timed automata as a formal model for discrete, distributed, timed systems. We develop a novel technique that constructs a complete, finite representation of such a system's state space. This representation is the complete, finite prefix of an unfolding in which concurrently enabled actions are partially ordered. We show that this technique is capable of reducing the size of the state space by magnitude. We are presently not aware of any state space reduction technique for timed automata with similar results.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Verifying Discrete, Distributed, Untimed Systems . . . . .	1
1.2	Introductory Example and Analysis of the Problem . . . . .	6
1.3	Existing Works and Yet Unsolved Problems . . . . .	8
1.3.1	Reduction Techniques for Distributed, Timed Systems . . . . .	8
1.3.2	Conclusions for a Novel Approach . . . . .	13
1.3.3	Netcharts . . . . .	14
1.3.4	Event Structures and Unfoldings . . . . .	15
1.4	A Novel Approach for Partial Order Semantics of Timed Systems and Unfoldings . . . . .	16
1.5	Notation . . . . .	17
<b>2</b>	<b>Networks of Message Passing Timed Automata</b>	<b>18</b>
2.1	Basic Definitions . . . . .	19
2.2	Classic Timed Automata . . . . .	19
2.3	Message Passing Timed Automata . . . . .	22
2.4	Global and Local Time Steps . . . . .	24
2.5	Reference Clocks and Time Stamping Messages . . . . .	24
2.6	Local Clock Semantics . . . . .	28
2.7	Region Abstraction of Timed Automata . . . . .	30
2.8	Timed-Stamped Region Semantics of Networks of Message Passing Timed Automata . . . . .	31
2.8.1	Time Stamping Messages with Zones . . . . .	37
2.9	Local-Region Semantics . . . . .	38
2.9.1	Independence in the Local Region Graph . . . . .	40
2.9.2	Event Structure for Local-Region Semantics of MTA <sup>+</sup> . . . . .	43
2.10	Summary of Networks of Timed Automata . . . . .	46
<b>3</b>	<b>Local Region Automata are equivalent to Timed-Stamped Networks of Timed Automata</b>	<b>48</b>
<b>4</b>	<b>Event Structures of Local Region Automata have a Finite, Complete Prefix</b>	<b>51</b>
4.1	Region Equivalence and Reference Clocks . . . . .	51
4.2	State Equivalence and Finite Representability of Region-Stamped Automata . . . . .	55
4.3	A Finite, Complete Prefix for Progressing MTA . . . . .	57
4.4	Constructing a Complete Finite Prefix . . . . .	59
<b>5</b>	<b>Evaluation of the Reduction Method</b>	<b>62</b>
5.1	A progressing MTA . . . . .	62
5.2	The State Space . . . . .	63
5.3	A Finite Complete Prefix of its Unfolding . . . . .	65
5.4	Analysis of the Prefix . . . . .	71

<b>6 Summary, Conclusion and Future Work</b>	<b>76</b>
<b>Index</b>	<b>81</b>
<b>List of Figures</b>	<b>83</b>
<b>References</b>	<b>84</b>
<b>Acknowledgements</b>	<b>87</b>
<b>Erklärung</b>	<b>89</b>

# 1 Introduction

The progress of time plays a significant role in many of today's computer-related systems: timing constraints might need to be satisfied to guarantee a correct behaviour, or timing information may be part of a system's specification. Controllers of industrial systems, protocols and asynchronous circuits are examples of such systems. Depending on the domain of application, it may be imperative to guarantee a failure-free operation of such a system under any circumstances. Achieving such a guarantee turns out to be especially difficult if the system of concern consists of several, separated components.

In this thesis we will consider discrete, distributed, timed systems. A system is *distributed* if it consists of several, separated components. It is *timed* if the progress of time influences how the system behaves. A system is *untimed* if the opposite is true. We say that a system is *discrete* if a part of its state is given by non-continuous values and if the system's dynamics can change these values discontinuously only. Such a discontinuous change is called a *discrete step* and can be conceived as being caused by an *action* of the system, e.g. pushing a button or the occurrence of a time-out event.

A popular and successfully applied approach to guarantee the *correctness* of discrete systems in general, and of discrete, timed systems in particular, is to *verify* the correctness through *model checking* [EMCGP99]. This computer-based technique is used to exhaustively examine the system's entire state space, which raises a problem in distributed systems: actions in different components that are not causally dependent on each other cause an exponential growth of the state space, a phenomenon known as *state space explosion* which greatly limits the applicability of the method. For untimed systems, various reduction techniques, like *stubborn sets* [Val91] or *unfoldings* [McM92], are known to alleviate the state space explosion. Unfortunately, these reduction techniques cannot directly be applied to timed systems. This raises demand for a specific solution for timed systems, which – to our knowledge – has not been provided yet.

Throughout this thesis we will define a novel reduction technique based on unfoldings for discrete, distributed, timed systems. The main result of the thesis is that our technique yields an equivalent representation of the system's state space which has significantly less memory requirements compared to established model checking techniques. We show that the result cannot hold for arbitrary discrete, distributed, timed systems but for a special class of these systems which we call *networks of message passing timed automata*. The thesis provides the theoretical foundations for an algorithmic solution, that itself will be left as future work.

## 1.1 Verifying Discrete, Distributed, Untimed Systems

We first want to introduce the idea of verifying distributed systems through model checking by considering untimed systems only. This makes it easier to highlight the related problems and successful solutions. We will change the context to timed systems in the next section.

Usually, a state of a distributed system can be partitioned such that each component has its own part of the state. A component *takes part* in a state change if its part of the state changes then. Hence an action belongs to a component if the component takes part in a step due to this action. Typically, a component in a distributed system is *sequential* which means that its discrete steps are totally ordered by causality. This ordering, of course, carries over to the component's actions and we also say that these actions are *dependent*, or *causally ordered*.

Two actions are *unordered*, if they are not causally ordered. Assuming sequential components in the following, this can only be the case if the actions belong to different components; but actions in different components may be causally ordered – a standard example is that receiving a message causally depends on the sending of that message.

Unordered actions can be executed in any order (or even together). One reason for unorderedness is concurrency. Two actions are *concurrent* if neither is cause for the other, or put differently, if neither requires the result of the other. Unordered actions may be *independent* which means that, independently of the order in which the actions occur, the same state is reached.

A sequential representation of a distributed system can only simulate unorderedness of actions by showing any possible order of these actions. As an example consider the two finite automata  $A_1$  and  $A_2$  in Fig. 1(a). The two actions  $a$  and  $b$  are concurrent and even independent. The (sequential) state space in Fig. 1(b) of these automata puts actions  $a$  and  $b$  into any order in which they may occur, which is called *interleaving* of  $a$  and  $b$ .

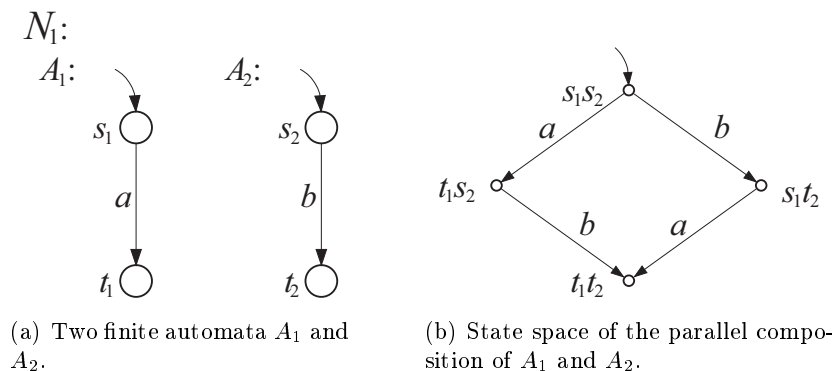


Figure 1: Unordered Actions

If one is interested in the correctness of a system, that is, if the system shall satisfy a set of (then to be defined) relevant properties – for instance whether the state  $t_1t_2$  is reachable in the parallel composition of  $A_1$  and  $A_2$  of Fig. 1(a) – one has two principle ways to *ensure* correctness: either the system is *constructed* to be correct, by proving the correctness as the system is specified and implemented, or the system is *verified* once it (or a model of it) has been created. Concentrating on the second approach, research and literature offer several ways of verifying that in a given system a certain property holds.

The widely-used approach of model checking exhaustively examines the system’s entire state space [EMCGP99]. Thereby, in order to prove that a property holds, it might become necessary to construct larger parts or even the entire state space of the system. Due to its sheer size and being a computationally simple task this is performed using computers. This computer-aided verification technique requires that the state space of the system to be verified is stored in the main memory of the verifying computer. The successful application of the method heavily relies on whether the state space can completely be represented in the main memory. If it does not fit in, then the examination might not be complete. Since the main memory limitation is strict, one is willing to spend more computing time if the state space can be represented more compact while preserving the kind of property one aims to verify.

There are several reasons why the state space becomes too big for a successful verification: for instance a large system itself is likely to have a large state space; a very compact, implicit representation of the state space in the system description may yield a large state space although the system description is small; and unordered actions in distributed systems may do so as well. Similarly there are several ways to achieve a more compact representation of the state space, each of them exploiting a specific structure in the state space. We will focus concurrency in distributed systems as a reason.

The major reason for a large state space in distributed systems are unordered actions. Through ordering such actions in any possible order to simulate their unorderedness, the state space grows exponentially in the number of unordered actions. In Fig. 2 we extend the sample system of Fig. 1 by a third automaton  $A_3$  and an additional action  $d$  which occurs in  $A_1$  and  $A_3$ . One model of composition to which we will refer later again is *synchronization on common actions*: action  $d$  occurs in  $A_1$  iff action  $d$  occurs in  $A_3$  –  $A_1$  and  $A_3$  *synchronize* on  $d$ . We show the thereby composed system’s state space in Fig. 2(b). If a set of automata is composed such that the behaviour of one automaton influences the behaviour of another automaton, for instance by synchronization, we conceive the set of automata as a *network*; the composition of  $A_1$ ,  $A_2$  and  $A_3$  yields the network  $N_2$  in Fig. 2(a). Together, they constitute a discrete, distributed system; each automaton is a component of the network. The network structure is given implicitly by the single automata as nodes and their common actions as edges.

Once can easily see that the state space is significantly larger than the system description: For  $n$  pairwise unordered events that are enabled in the same state, we will say *concurrently enabled* events, the state space contains  $2^n$  states and  $n!$  different runs from the common initial state to the common terminal state. This phenomenon is known as state space explosion. The state space explosion can be countered in several ways, the two most prominent approaches are *stubborn sets* and *unfoldings*.

The first approach, stubborn sets, exploits that interleavings of independent actions generate redundant information in the state space. If one is interested in the common final state and not in the intermediate states, where just some of the independent actions have occurred, it is sufficient to explore just one interleaving. As an example consider Fig. 3(a) where all runs lead to the terminal state  $u_1t_2u_3$  and hence it is sufficient to follow the action sequence  $abcd$  if one is interested in the reachability of the terminal state.

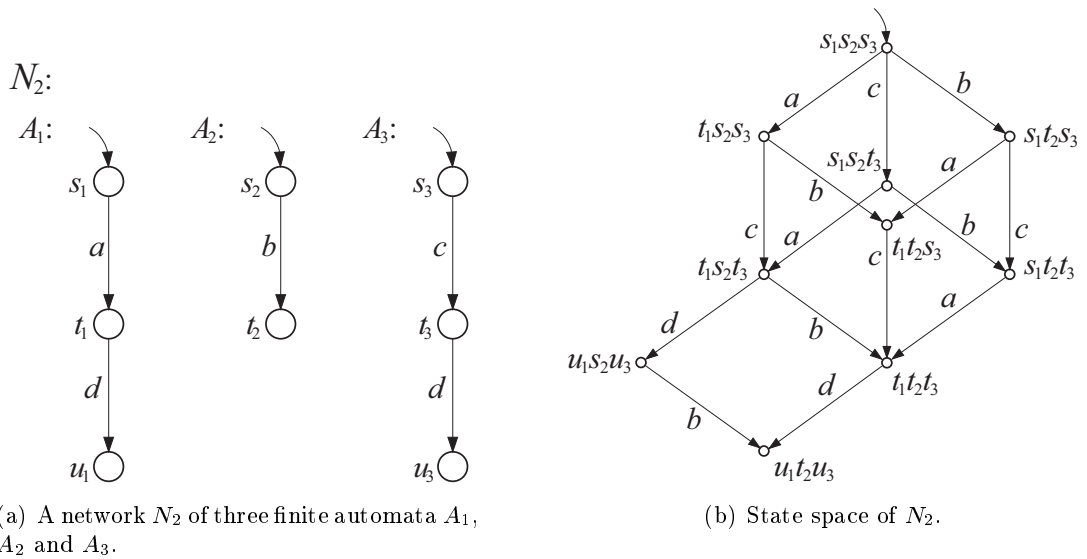


Figure 2: State Space Explosion

Valmari has shown in [Val91] that by exploring only a subset of enabled actions in a given state, the *stubborn set*, satisfiability of linear temporal logic formulas without next-state operator is preserved while the states explored in total is no longer exponential in the number of concurrent actions. Roughly speaking, if an action  $a$  belongs to a stubborn set  $S$ , then all enabled actions that depend on  $a$  (or that  $a$  depends on) belong to  $S$  as well – leaving out all independent actions. In Fig. 3(a), one stubborn set of  $s_1s_2s_3$  is  $\{a\}$ . Valmari applied stubborn sets for model checking Petri nets. The quite similar idea of *ample sets* which differs in details only was published by Peled [Pel94] and is widely used in the verification of automata. In the optimal case, the reduced state space contains exactly one linearization per set of concurrently enabled actions, which is linear in the size of the system.

A different approach to reduce the required memory is simply not to do the “mistake” of ordering unordered events but to represent their unorderedness directly. This idea is the basis for the *unfolding* technique which was (again) applied in the verification of Petri nets first. A Petri net describes concurrent behaviour directly in its structure: unordered actions are structurally not ordered in the system description [Rei85]. By “unfolding” the system description, one can construct a representation of a Petri net’s state space in which concurrent actions are partially ordered [NPW79]. Because in unfoldings of Petri nets, unordered actions are not ordered arbitrarily but remain unordered, the state space explosion simply does not occur.

By conceiving a single finite automaton as a special Petri net (where all actions are in a total causal order), the unfolding technique of Petri nets becomes applicable to networks of automata as well. Unlike in the sequential state space, a state of the system is partitioned into the “local” states of its components, each called a *condition*, represented as a

Petri net place in Fig. 3(b). Actions usually do not modify the states of all components, hence – in this representation – the execution of an action reads its *pre-conditions* and generates corresponding *post-conditions* that represent just the changed part of the state while the rest remains put. Such an execution may be conceived as a “local” step and is called *event*, represented as a Petri net transition in Fig. 3(b). By combining conditions one can construct a global state of the system. Due to the partitioning, concurrent events in different components have disjoint pre- and postconditions. They can be represented in an unordered way, or more precisely, all events of the system are partially ordered instead of totally ordered like in the state space. Ideally, an unfolding’s grows linear in the size of the system [ERV96].

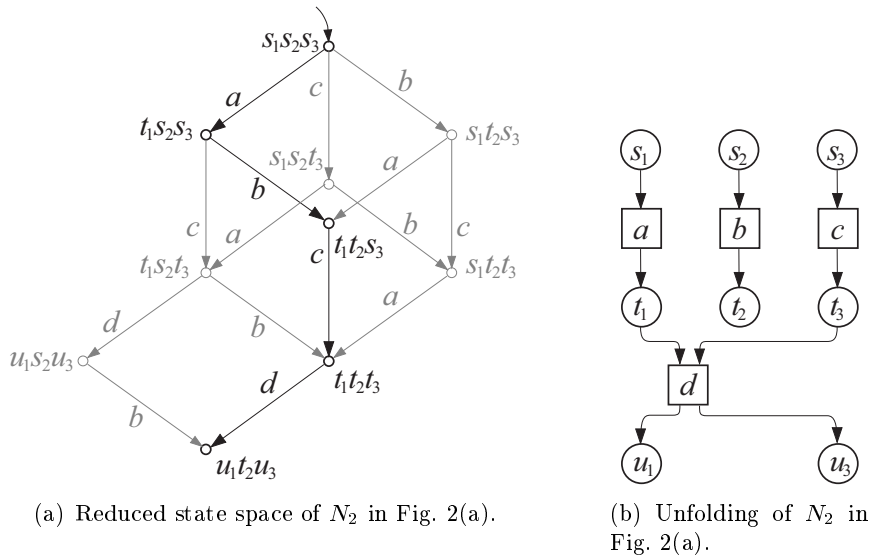


Figure 3: Unfolding

The construction of the unfolding as well as the use of stubborn sets heavily relies on a proper notion of independent actions that can be executed in arbitrary order and yield the same final state. We assume that a single finite automaton describes a strictly sequential system, i.e. the automaton itself does not simulate concurrent behaviour – the corresponding syntactic criterion is that in each automaton, an action occurs at most once. With this reasonable assumption, in networks of finite automata any two actions are independent if and only if they occur in disjoint sets of components. One can apply a structural property (different components) to obtain information about a semantic property (independence).

In the next section we show that such a nice structural property is not straight forward for timed systems. The lack of a simple structural property renders the known state space reduction techniques unapplicable to the verification of timed systems in first place. Throughout the thesis we will scrutinize how to obtain a similar structural property for timed systems that allows us to apply the technique of unfoldings for distributed, timed systems.

## 1.2 Introductory Example and Analysis of the Problem

In the previous section we explained verification and reduction techniques for discrete, distributed, untimed systems. We will now turn to timed systems and observe the differences. We will present a fairly standard example<sup>1</sup> that demonstrates the difficulties in verifying timed systems in comparison to untimed systems in the following. But we need to introduce the notion of time into the system first. There exist various ways to model discrete, distributed, timed systems, one of the most accepted models are timed automata.

**Timed Automata** Alur and Dill introduced the fairly elegant model of *timed automata* [AD90, AD94] which is now a widely accepted way to model real-time systems, or, more general, systems with continuous time. The established concept of a finite automaton is extended with a notion of continuous-time clock variables that may be reset by a transition which is guarded by clock constraints. Furthermore, states may impose invariants in the form of clock constraints. Clock variables advance in time simultaneously, invariants and guards restrict the possible timed behaviour of the system.

As an example, have a look at the timed automaton  $TA_1$  in Fig. 4(a). Like the finite automaton  $A_1$  in Fig. 1(a) it has an initial state  $s_1$  and another state  $t_1$  which can be reached through action  $a$ . Furthermore,  $A_1$  has the *clock variable*  $x$  which is reset to 0 whenever  $a$  occurs in state  $s_1$  (denoted by the assignment  $x := 0$ ). A state of  $TA_1$  consists of a discrete state, say  $s_1$ , and the values of all clock variables, called *clock valuation*. Additionally to a discrete step due to action  $a$ , while being in  $s_1$ , time may elapse by some (real-valued) amount  $d$  which increases the value of all clock variables in the system by  $d$  and yields another state. This is called a *time step*; we will label time steps with the symbol  $\delta$  throughout the thesis.

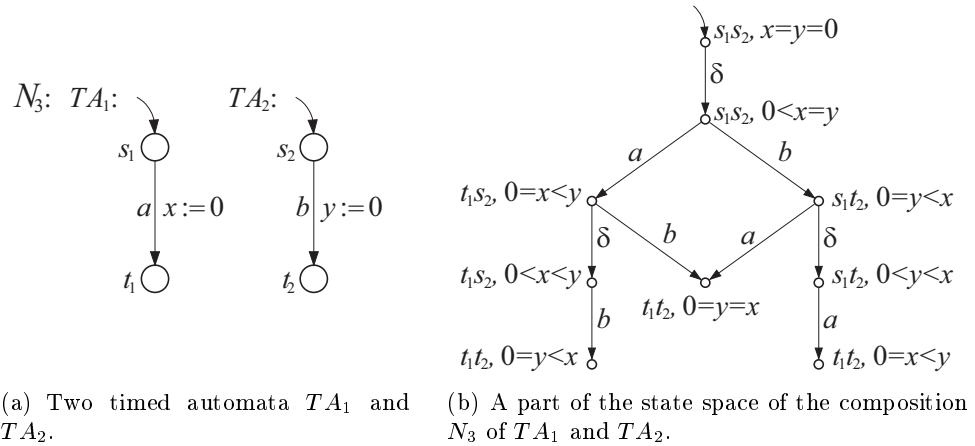


Figure 4: Commuting and Non-Commuting Actions in Timed Systems

Similar to finite automata, several timed automata may be composed. The most

<sup>1</sup>This example was, for instance, given before in [BJLY98].

common mode of composition yields *networks of timed automata* (NTA) that synchronize on common actions, where an action can occur in one automaton if and only if it occurs in all other automata that have this action. But more importantly, a time step in a NTA occurs globally: the progress of time increases the value of *all* clock variables in the system by the same amount; therefore the components of a NTA synchronize on the progress of time as well!

A timed automaton's underlying transition system is of infinite size since a state includes the values of all clock variables which are real numbers. Reasonably assuming that the description of a timed automaton uses rational numbers only, the transition can equivalently be reduced to a finite model. The basic idea in the equivalent reduction is to represent infinite sets of clock valuations symbolically, in which all clock valuations in the set yield equivalent<sup>2</sup> behaviour. The first reduced model is the *region automaton*, introduced in [ACD90] which is exponential in the number of clock variables and the largest constant in the system [AD94]. A second way to obtain a finite model is to construct the *zone automaton* which operates on clock constraints, i.e. (in)equations on differences of clock variables [Dil89]. Despite having a similar worst-case complexity compared to the region automaton, the zone automaton has been proven to be quite efficient in verifying industrial systems, see for instance [Feh99], [HSL97], and [TY01].

**Concurrent Actions are not Independent** If we consider multiple timed automata together, like  $TA_1$  and  $TA_2$  in Fig. 4(a), they constitute a composed system where each automaton is sequential component.

Just as for finite automata, any two actions enabled at the same time are concurrent iff they belong to different components of the system. For instance, actions  $a$  and  $b$  are concurrent in the composition of  $TA_1$  and  $TA_2$ . But in contrast to finite automata, the state space of the composed system shows that  $a$  and  $b$  are not independent.

In Fig. 4(b), we depict a part of a finite, abstract representation of the composed system's state space. Letting some time pass (synchronously on all clock variables of the system) in the initial state  $s_1s_2$  yields a state where  $x$  and  $y$  are equal but larger than 0. The occurrence of  $a$  then resets  $x$  to 0 while  $y$  is left untouched which yields a state  $(t_1s_2, 0 = x < y)$ . If no time passes before  $b$  occurs,  $y$  is reset and the state  $(t_1t_2, x = 0 = y)$  is reached. Just as in finite automata, the sequence first  $b$  then  $a$  (without time in between) yields the same state.

If instead in  $(t_1s_2, 0 = x, 0 < y)$ , we let some time pass,  $x$  gets a value larger than 0 but less than the value of  $y$  because  $y$  is increased by the same amount. We reach  $(t_1s_2, 0 < x < y)$ . The occurrence of  $b$  now results in another state  $(t_1t_2, 0 = y < x)$  which is different from  $(t_1t_2, x = 0 = y)$ . Similarly, the occurrence of  $b$ , then letting some time pass, followed by the occurrence of  $a$  yields  $(t_1t_2, 0 = x < y)$  which is different from the other states as well.

We see that despite their concurrency, different interleavings of  $a$  and  $b$  yield different states. Hence  $a$  and  $b$  in  $TA_1$  and  $TA_2$  of Fig. 4(a) are not independent and it is not sufficient to explore just one interleaving if one is interested in preserving those states

---

<sup>2</sup>precisely: bisimilar

which are reachable by firing all concurrent actions. This example shows that the simple criterion for independent actions in networks of finite automata, is not applicable in NTA to define a correct reduction technique.

**The Problem** We just have seen that despite the state space explosion (which is even worse than in finite automata) due to concurrent actions, the known partial order reduction techniques that use stubborn sets seem not to be feasible to reduce the state space. The promising approach of unfoldings, which avoid interleaving unordered actions, require a proper notion of independence for the notion of local state and local step. But the known structural criterion for independence does no longer hold.

The open question is, how the state space of a discrete, distributed, timed system can be reduced to alleviate the state space explosion while the property of interest (being reachability of terminal states at least) is preserved.

### 1.3 Existing Works and Yet Unsolved Problems

The problem, as we have just stated it, is not new. Various research has been done in the area of reduction techniques for distributed, timed systems to counter the state space explosion. We will present the most prominent results in the following where we will put a focus on timed automata. Up to now, we are not aware of any reduction technique that has been proven to be effective in case studies. We will therefore give a profound analysis of one of the most promising approaches in this field and we will eventually show that there is still work to be done.

Through our analysis we will find that some structural properties of networks of timed automata actually influences the applicability of a reduction method. Taking our profit from this insight, we will have a look on a certain class of distributed, untimed systems with promising properties for our purpose.

#### 1.3.1 Reduction Techniques for Distributed, Timed Systems

**Time Petri nets** Similar to timed automata, *time Petri nets* make the enabledness of transitions dependent on the progress of time. Literature knows various classes of time Petri nets, the most prominent associating with each transition an earliest and latest firing time since the transition became enabled last [MF76], [Sta90]. For this class of time Petri nets, Yoneda et al. defined a state space reduction technique based on stubborn sets. They show that satisfiability of a timed temporal logic is preserved, and demonstrated its effectiveness in experimental results [YS97]. Lilius improved the technique in [Lil98] regarding memory consumption and shows how to efficiently explore the state space using unfoldings. Fleischhack and Stehno defined an unfolding for time Petri nets by modelling the progress of time as part of the net structure and then showed how to effectively compute a complete finite prefix. [FS02]

However the model of earliest and latest firing times of transitions is less expressive than timed automata [Min99a]. Therefore the results for timed Petri nets cannot be applied to timed automata directly.

**Independence in Timed Automata** A first work on independent actions in timed automata is by Pagani [Pag96, Pag97]. She shows that in many cases actions become dependent because of timing which greatly reduces the amount of interleaved steps of independent actions in the automaton’s state space. The set of dependent actions in a NTA can be characterized as follows [Min99b]. Let  $a$  and  $b$  two distinct actions in different timed automata and let  $R_a$  and  $R_b$  the sets of clock variables that are reset upon the occurrence of  $a$  and  $b$  respectively. Actions  $a$  and  $b$  are dependent if one of the following cases is met:

1. Progress of time enables  $a$  but disables  $b$ .
2. Progress of time enables  $a$  but  $b$  disables progress of time.
3.  $R_a \neq \emptyset$  and  $R_b \neq \emptyset$ .
4.  $R_a \neq \emptyset$  and progress of time enables  $b$ .
5.  $R_a \neq \emptyset$  and progress of time disables  $b$ .
6.  $R_a \neq \emptyset$  and  $b$  disables progress of time.

In cases 1 and 2, the enabledness-condition of independent actions is violated. If cases 3 to 6 hold, actions  $a$  and  $b$  are not commuting.

Based on this characterization, Pagani proposes a partial order reduction technique for timed automata based on ample sets. Dams et al. improve her results by introducing the asymmetric notion of *covering* actions [DGKK98]. An action  $b$  covers an action  $a$  if the set of timed states reached by executing first  $a$  and then  $b$  is a subset of the timed states reached by executing first  $b$  and then  $a$ . The corresponding method uses ample sets and allows to follow the covering action and skip the covered action without omitting a reachable state.

An important meta-result about partial order reduction techniques for timed automata has recently been published by Salah et al. [SBM06]. They have shown that the union of all clock zones that are reachable by different interleavings of the same set of concurrently enabled actions is convex. Furthermore, they proved that this union is *not* an over-approximation, but describes *exactly* the set of reachable clock valuations.

**Partial Ordered Semantics** The previously mentioned approaches of Pagani and Dams aim at identifying the independence that is present in a timed automaton’s state space in order to apply partial order reduction techniques. As we have already seen in Sect. 1.2 the global progress of time introduces dependencies between otherwise independent actions. The criterion of independence cannot be checked syntactically, but requires to consider the semantics of the system.

Therefore Bengtsson et al. – considering networks of timed automata that synchronize on common actions – suggested to desynchronize the progress of time by allowing time to elapse in each component separately [BJLY98]. Their aim is that the new semantics, allows a simpler, syntactical criterion for independence of actions.

Their “local-clock semantics” introduces local time steps of a component that increase the clock variables of this component only while all other components’ variables stay put. Discrete steps due to actions are as in standard semantics of timed automata. It then follows that two actions are independent if and only if they belong to different components. However, since different amount of time may elapse in different components, the local-clock semantics allows to reach states which haven’t been reachable before. Worse, two components could synchronize on a common action although different amounts of time have elapsed, thereby allowing runs which have not been possible before.

In normal semantics, the network of timed automata  $N_4$  that is shown in Fig. 5 cannot reach the state  $t_1t_2$  since the synchronizing action  $c$  is never enabled:  $c$  becomes enabled in  $TA_1$  if exactly 1 unit of time has elapsed (on  $x$  and  $y$  simultaneously), while  $c$  is enabled in  $TA_2$  at 2 time units only due to the lack of clock resets for  $x$  and  $y$ . Though,

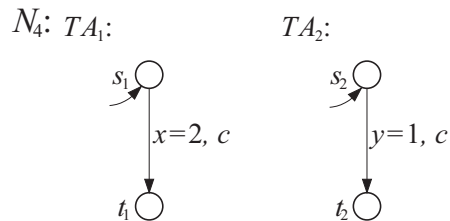


Figure 5: Dependent Actions in the Zone Automaton.

in local-time semantics, we can let pass 1 unit of time in  $TA_1$  and, independently, 2 units of time in  $TA_2$  thus reaching a state where  $c$  becomes enabled and  $t_1t_2$  is reachable.

To prevent this illegal behaviour, Bengtsson et al. introduce a “reference clock” into each component. A reference clock does not occur in a guard or invariant and is never reset. Hence it measures the total time that has elapsed in each component in a run. The two components may synchronize only if their reference clocks are equal, i.e. if they have spent the same amount of global time. In the example of Fig. 5, this restriction prevents  $c$  from becoming enabled in local-time semantics.

Bengtsson et al. then claim that the set of locally reachable states in which the same amount of total time has elapsed in every component, is exactly the set of states reachable by standard semantics. They furthermore claim that this result extends to the zone automaton. Based on this claim and the re-established independence relation they define a partial order reduction technique on the zone automaton using ample sets which is claimed to preserve reachability.

In a subsequent work, Minea provides proofs for the equivalence of local-time semantics and standard semantics in the non-abstract (hence infinite) representation [Min99a], [Min99b]. He also provides proofs for a finite representability of the state space despite the presence of unbounded reference clocks and suggests a slightly different definition of the ample set method to reduce the state space.

Despite the promising result, neither Bengtsson et al. nor Minea present a case study or validate the method experimentally. A closer and profound analysis of the works reveals that although the suggested reduction method is correct, it makes no use of the

“new” independence relation – their reduction technique does not follow their initial idea. Unfortunately, it remains unclear whether the idea of a partial order semantics for timed automata actually yields a reduction technique for timed automata.

**Problems with Partial Ordered Semantics** The suggested beauty of the result of Bengtsson and Minea is that in local-time semantics two steps “are independent if they correspond to disjoint [actions] in different processes”<sup>3</sup>. This holds for the representation of time by real numbers but ceases to hold in the zone abstraction: the local zone-based semantics still represents the clock valuations by a single zone on all clocks which spans over all components.<sup>4</sup> This implies that clock constraints between clocks variables of different components are part of the state. Then on this data-structure, resetting clocks no longer renders actions independent because constraints between clocks of different components (which are part of the state) are influenced. As an example consider the zone automaton of the timed automata  $TA_1, TA_2$  of Fig. 4(a) according to the local semantics of [BJLY98]. In fact, the automaton in this special case does not differ from the classical zone automaton.

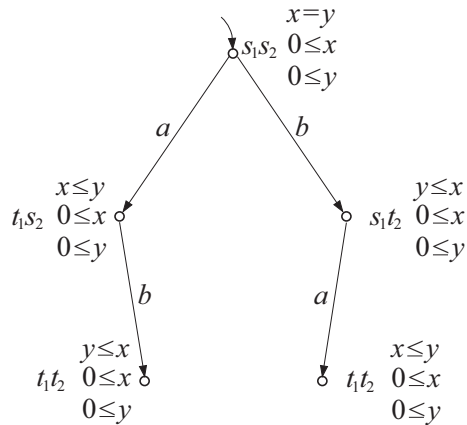


Figure 6: Dependent Actions in the Zone Automaton.

In the initial state, where both clocks  $x$  and  $y$  have initially equal values, and both clocks can advance independently in time to values greater than or equal to 0. The occurrence of  $a$  resets  $x$  to 0 thus being less than or equal to  $y$  from whereon  $x$  can advance locally again. Similarly after the occurrence of  $b$  where relations between  $x$  and  $y$  are inverse. Just as in Fig. 4(b), we obtain different states for different interleavings; Figure 6 depicts the entire zone automaton. We can see that actions  $a$  and  $b$  are not independent in the zone automaton, while Bengtsson et al. state the opposite.

Minea eventually realizes that the structural independence relation is not sufficient for semantic independence in the local zone semantics. After presenting an example, he observes that:

<sup>3</sup>cf. Abstract on p.485, [BJLY98]

<sup>4</sup>Sect.3.1, p.493, [BJLY98] and Sect.3.6.1, p.59, [Min99b]

Exploring either of  $\xrightarrow{a}$  and  $\xrightarrow{b}$  restricts the current local-time zone to a fragment where the other transition is no longer enabled. Thus, even though  $\xrightarrow{a}$  and  $\xrightarrow{b}$  are independent, selecting only one of them as an ample set would violate condition **C0**.<sup>5</sup>

The mentioned condition **C0** has to be satisfied by the ample set  $ample(s)$  of a state  $s$  and is defined as:  $ample(s) = \emptyset$  iff  $enabled(s) = \emptyset$ . The necessity to include “structurally independent” actions  $a$  and  $b$  in the ample set to satisfy condition **C0** leads to ignoring the idea of using an independence criterion to explore just a single interleaving to reach a common successor state.

Minea then strengthens the conditions for the ample sets to yield correct results. However, the suggested independence relation is not used to define a partial order reduction technique for networks of timed automata; it remains unclear how the structural criterion for independent actions contributes to the reduction method.

### More Results from Timed Automata and Their Relation to Reduction Methods

There is another important property of the reduction method of Bengtsson et al. It seems to be doubtful that, even if the independence relation held in the zone abstraction, we would obtain a finite representation of the local-time semantics for networks of timed automata with synchronization on common actions.

Remember that for each finite automaton the set of sequences of actions that is accepted by the automaton and leads to a terminal state is called the automaton’s *language*. The language of a finite automaton is *regular*. Likewise in timed automata, when assuming that every state is terminal, the *untimed language* of a timed automaton is regular and is exactly the language that is accepted by the region automaton [AD94]. This result extends to networks of timed automata with synchronization on common actions since those have a finite region automaton as well.

Unfortunately, from trace theory follows that regular languages are not closed under commutations of independent actions in general. Depending on the language and the choice of the independence relation, commuting actions may yield a language that can no longer be accepted by a finite automaton [Die95, pp.3–42,167–204]. Consider the following small network  $N_5$  of timed automata in Fig. 7.

In  $TA_1$ , the invariant  $x \leq 1$  prevents the system to stay more than one time-unit in  $s_1$  without performing an action. The guard  $x = 1$  restricts the firing times of  $a$  and  $c$  to integral moments. Similarly for  $TA_2$ .

The runs of  $N_5$  in the usual semantics of timed automata would accept the untimed regular language  $((ab)|(ba))^+c$ . When applying local-clock semantics both automata can accept an arbitrary number of  $as$  and  $bs$ , respectively, before accepting a  $c$  together. However due to the requirement that in both automata the same amount of global time has to elapse, the number of actions  $a$  and  $b$  has to be equal. For a word  $w$  let  $|w|_a$  denote the number of occurrences of  $a$  in  $w$ . In local-clock semantics,  $N_5$  accepts the

<sup>5</sup>Sect.3.9, p.70, [Min99b]; The events  $a$  and  $b$  are not related to the example of Fig. 6.

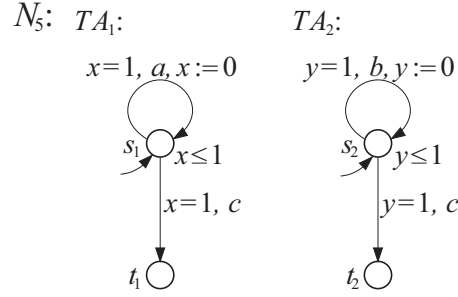


Figure 7: Network  $N_5$  of Timed Automata  $TA_1, TA_2$  with non-local choice.

language  $\{w \mid w = w'c \text{ with } |w'|_a = |w'|_b\}$  which is known to be *not regular*. We can formulate the following result:

**Lemma 1:** *Let  $L$  be the untimed language that is accepted by a network  $N$  of timed automata with synchronization on common actions. Let  $I$  be the independence relation on the actions of  $N$  with  $(a, b) \in I$  iff  $a$  and  $b$  occur in disjoint sets of components of  $N$ . Then the trace-closure of  $L$  under independence relation  $I$  is not regular. \**

From Lemma 1 follows that even the region- or zone representation of the state space in local-clock semantics is not finite and we can conclude that the approach of local-clock semantics by Bengtsson et al. cannot yield an applicable partial order reduction method for NTA that synchronize on common actions.

### 1.3.2 Conclusions for a Novel Approach

We will draw two conclusions from our analysis of the existing approaches above. First of all, we have seen that the chosen class of timed systems in combination with the independence relation may yield practically unusable results. Therefore great care must be taken when choosing these two ingredients. Secondly, the notion of a global state that cannot be partitioned into the local states of the system's components may be incompatible with a simple structural criterion for independent actions.

We still think that the local-time approach is a promising one for it yields an elegant characterization of independence, at least in the case of a real-valued representation of clock values. What needs to be done is to find an abstract representation of a timed automaton's state space such that a state in this representation can be partitioned into the local states of the system's components. Furthermore, we need additional properties, for the structure of the system, that exclude a result similar to Lemma 1.

Regarding the structure we will have a brief look on *netcharts*, a special class of networks of (untimed) finite automata, that synchronize on messages instead of common actions, since this class has some interesting properties that will turn out to be useful for our reduction method. Our aim to achieve a representation of a partitionable abstract state gives rise to the idea of applying the *unfolding* method in order to succinctly

represent the state space of a distributed, timed system. We will therefore highlight some important results from these fields in the two subsequent sections.

### 1.3.3 Netcharts

The model of netcharts originates from the search for regular message-sequence chart languages (MSC languages) [HMKT00, HMKT99]. The proof of regularity of netcharts given in [MKT03] makes use of the conversion into  $B$ -bounded Petri nets (for some bound  $B \in \mathbb{N}$ ) and networks of finite automata that communicate over buffered message channels.

We may legitimately conceive a *netchart* as a set of finite automata where some actions are special *communicating actions*. For a given set of message types, a *send* and a directly corresponding *receive* action for each message type is provided. Each such action must occur exactly once in the entire system where corresponding send and receive actions must be in different components. The communicating actions thereby define a point-to-point communication channel which is meant to be a buffer, i.e. messages can be stored in the channel arbitrarily and are undistinguishable. Sending a message is always possible and puts a message in the corresponding buffer. A receiving action occurs only if the corresponding buffer is not empty, which then removes a message from the buffer. Hence corresponding send and receive actions depend on each other. Actions in different components are concurrent. A netchart is defined as *B-bounded* if in every reachable state, no message buffer holds more than  $B$  messages.

In Fig. 8 we show an example of a 1-bounded netchart and its equivalent representation as a Petri net.

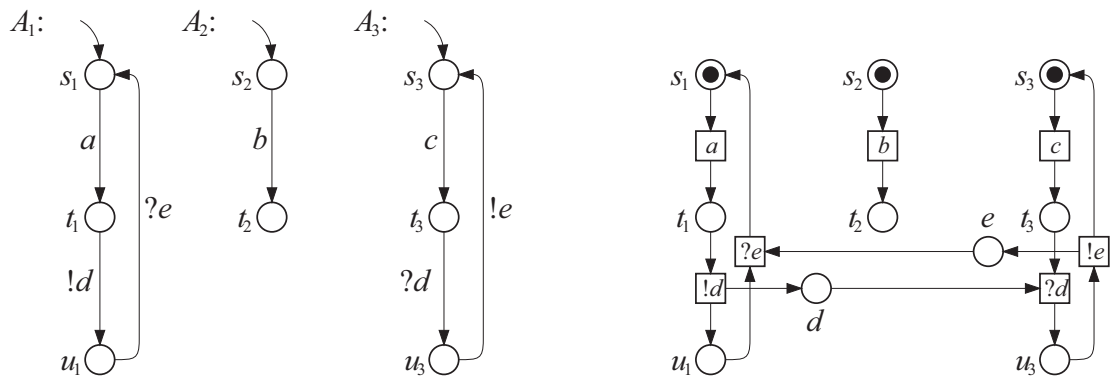


Figure 8: A netchart of three components  $A_1$ ,  $A_2$ , and  $A_3$  on the left and its equivalent representation as a Petri net on the right.

A beautiful result in netcharts is that any two concurrently enabled actions are independent. More explicitly, in a state, two enabled actions are independent iff they are in different components, and in case it is a pair of corresponding send and receive actions, there is at least one message in the buffer. The latter condition ensures that the receive

action can receive a message even if the send action has not sent yet.

More importantly for us, it follows that due to this independence relation, the language of a netchart is regular iff there exists a natural number  $B$  such that the netchart is  $B$ -bounded. [MKT03].

We take this nice result as an inspiration for a practically relevant class of distributed, timed systems for which an applicable reduction method is possible.

### 1.3.4 Event Structures and Unfoldings

The notion of the unfolding of a Petri net that we briefly introduced in Sect. 1.1 originates in the more general notion of an *event structure* that provides a set of *events* together with a *causality* relation, and a *conflict* relation over these events. Nielsen et al. suggested to conceive an *event* as a unique occurrence of a transition (or more general: as an occurrence of an action) in a run of the system [NPW79]. Two events are causally ordered if the occurrence of the dominant event's action is necessary for the occurrence of the dependent event's action. Two events are in conflict if the occurrence of either event's action excludes the occurrence of the other event's action.

This interpretation of an event structure is not limited to Petri nets. The occurrences of actions  $!d$  and  $?d$  in the netchart of Fig. 8 yield causally related events, the occurrences of  $a$  and  $c$  in  $TA_1$  of Fig. 7 yield conflicting events. Two events that are neither causally ordered nor conflicting are concurrent. With this interpretation it is possible to represent *all* runs of discrete system in an event structure without enforcing an order on concurrent events. A run of the system is described by a conflict-free downward-causally closed set of events, that is called *configuration*.

Usually, the events are *labelled* with the corresponding actions, which yields a labelled event structure. Any linearization of a configuration of a labelled event structure yields a run of the system. In case concurrent events are labelled with independent actions, all of these linearizations lead to the *same* state.

Cyclic systems may have infinite runs. Because an event represents a unique occurrence of an action, the event structure that describes the runs of the system may have infinitely many events. A model checking technique that uses unfoldings (or event structures) to represent the state space cannot operate on such an infinite structure.

The achievement of McMillan was to see that in systems with a finite state space, there exists a set  $Pref$  of events of the labelled event structure such that for that every reachable state  $s$  of the system,

- there exists a configuration  $K \subseteq Pref$  that leads to  $s$ , and
- for every action  $a$  that is enabled in  $s$ , there exists an event  $e \in Pref$  that extends  $K$  (i.e.  $K \cup \{e\}$  is a configuration).

A set of events  $Pref$  that satisfies the above properties is called a *complete, finite prefix of the unfolding* of the system. In Fig. 9, we depict an example of a prefix of an unfolding of the netchart in Fig. 8. McMillan has shown how to compute such a

prefix [McM92]. Because the prefix represents all reachable states of the system, it is a suitable for model checking. Due to their correspondence, we use the terms ‘unfolding’ and ‘event structure’ synonymously.

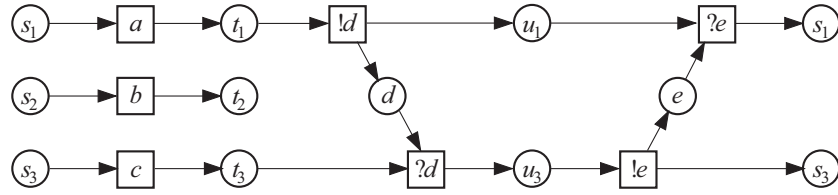


Figure 9: A prefix of the unfolding of the netchart in Fig. 8 in the graphical notation of a petri net.

#### 1.4 A Novel Approach for Partial Order Semantics of Timed Systems and Unfoldings

In the previous section we have looked into existing results for partial order reduction methods of timed systems and analyzed their weaknesses. Despite the results of our analysis, we still conceive the idea of desynchronizing the progress of time as convincing to obtain a partial order reduction method.

Basically, we are going to pick up the key idea of Bengtsson et al. to define a partial order semantics for a special class of timed automata, which can be conceived as *timed netcharts*, i.e. netcharts with time. We will define a semantics that finitely represents the state space of the system and enjoys the independence of concurrently enabled actions. This semantics will allow us to define an unfolding with the same expressivity and for which we can show the existence of a complete, finite prefix.

An outline of our approach shall be illustrated now.

Before the announced partial order semantics can be given, some intermediate steps are necessary. Firstly, of course, the “intended” global, concrete semantics of timed netcharts with real-valued clocks and a global progress of time is given. Then we define, in the spirit of Bengtsson et al., a local, concrete semantics that allows time to progress locally in each component and that gives rise to an independence relation on actions that is similar to the one for netcharts. In order to achieve a finite representation of the system’s state space, we have to define a reduced model either using the region automaton or the zone automaton. For reasons elaborated later on in Sect. 2.8.1, the abstraction to zones raises some difficulties we can avoid by using regions. Thus we will provide global, abstract semantics and local, abstract semantics with regions, each corresponding to their concrete counterparts. These definitions are given in Sect. 2.

We are going to be very careful when defining the abstraction that yields a finite representation of the system’s state space in local semantics in order to preserve the independence relation. The key idea to preserve independence is to use truly local states. In the local, abstract semantics each component will have its own state that

is structurally separated from the states of the other components. Then actions will modify the states of the involved components only and independence is kept.

We will then prove what the result of Salah et al (c.f. Sect. 1.3.1) suggests, that despite abstracting from concrete clock values and using constraints on clock valuations for each component only, the local, abstract semantics features exactly the same set of reachable states as the global, concrete semantics.

The local, abstract semantics is given by totally ordered runs, only. However, through our independence relation which also holds in local abstract semantics, we will then be able to define an event structure which gives us the notion of a partially ordered run. We then show that the partially ordered runs and the totally ordered runs are equivalent: each totally ordered run can be partially ordered and each partially ordered run can be totally ordered such that they yield the same state. Hence the partial order semantics is equivalent to the global semantics. This is going to be done in Sect. 3.

As the final step, we show in Sect. 4 that a finite prefix of the event structure is sufficient to represent all globally reachable states of the system. This result gives rise to a verification method that constructs the unfolding of the system directly and thereby alleviates the state space explosion problem.

A small exemplary case study in Sect. 5 will evaluate the size of the finite prefix that represents the state space which is followed by a conclusion about our approach in Sect. 6.

## 1.5 Notation

We close this section with a brief introduction into our notation and auxiliary mathematical tools which we will need throughout the thesis.

By  $\mathbb{N}$ ,  $\mathbb{Z}$ , and  $\mathbb{R}$  ( $\mathbb{R}_{\geq 0}$ ) we denote the set of natural numbers, integers, and (non-negative) real numbers, respectively.

**Sets** Let  $S, T$  be sets. With  $S \uplus T$  we denote the disjoint union of  $S$  and  $T$  which is defined only if  $S \cap T = \emptyset$ . By  $2^S$  we denote the power set of  $S$ .

**Multi-sets** Let  $U$  be a set. By  $\mathcal{M}(U)$  we denote the set of all multi-sets over  $U$ , where  $\square$  is the empty multi-set. Let  $m \in \mathcal{M}(U)$  and  $u \in U$ . We denote the multi-set addition of singleton multi-sets  $[u]$  by  $m + u$  and the corresponding difference by  $m - u$ .

**Tuples and Vectors** Let  $S_1, \dots, S_n$  be sets. We use bold-face symbols to denote elements of their cross product:  $\mathbf{s} \in S_1 \times \dots \times S_n$ .

**Sequences** We use  $\sqsubseteq$  to denote the prefix-relation on sequences: for sequences  $\rho_1, \rho_2$ ,  $\rho_1 \sqsubseteq \rho_2$  holds iff there exists a (possibly empty) sequence  $\rho'_1$  such that  $\rho_1 \rho'_1 = \rho_2$ . If  $\rho$  is a sequence over elements of  $U$ , then for any  $a$ , by  $|\rho|_a$  we denote the number of occurrences of  $a$  in  $\rho$ .

## 2 Networks of Message Passing Timed Automata

This section is dedicated entirely to the definition of networks of message passing timed automata (MTA) and its semantics. MTA are sets of timed automata that communicate by exchanging messages asynchronously like netcharts (see Sect. 1.3.3). The aim of this section is the definition of a partial order semantics for this class of timed automata such that any two concurrently enabled actions are independent. Based on these semantics, we can formulate an event structure where concurrent events are partially ordered. In the next section 3 we prove that this event structure is equivalent to the state space in conventional, sequential semantics of timed automata.

To reach our goal in this section, some intermediate steps are necessary that facilitate proofs and provide a better understanding of the system and its behaviour.

We repeat the formal definition of *timed automata* (TA) and their semantics in terms of the underlying transition system. From thereon we can define *networks of message passing timed automata* and their semantics. Being an extension of TA, the semantics of MTA represents time by real numbers leading to an infinite state space, and it lets time advance in all components simultaneously; we refer to this semantics as *concrete, global semantics*.

Firstly, we need to show that the approach of Bengtsson et al. [BJLY98] to desynchronize the progress of time componentwise is applicable to our class of networks of timed automata. We are going to define *concrete, local semantics* of MTA, still having a real-valued notion of time but where time may advance locally in a component. As in [BJLY98], it is necessary to introduce reference clocks and, by the chosen mode of synchronization, to time-stamp messages. The latter ensures that no message can be received before it is sent to prevent non-causal runs. To facilitate proofs of correctness, we refine the concrete, global semantics stepwise by introducing reference clocks first (concrete, global semantics *with reference clocks*) followed by the time-stamping (*time-stamped, global semantics*) before the progress of time becomes desynchronized. We show that these semantics are bisimilar wrt *synchronized* states in which the same amount of time has elapsed in each component. In Fig. 10 we depict an overview of the semantics and their relations.

Our second step is to show that the idea of desynchronized progress of time can be applied if time is not represented by concrete values, but some abstract representation. We will define *abstract, local semantics* of MTA. We abstract clock valuations by the *region equivalence* which we explain for TA first and then show that they can be applied to MTA in a straight-forward way. Similar to the concrete semantics, we refine the semantics step-wise by introducing reference clocks first (abstract, global semantics with reference clocks), before we time-stamp messages with clock regions (*region-stamped, global semantics*) to finally desynchronize the progress of time. In the next section 3 we prove the bisimilarity of abstract, local semantics to concrete, global semantics. Each of these abstract semantics relates directly to its concrete counterpart through the region abstraction of timed automata. This region abstraction establishes a *time-abstract bisimulation relation* which yields the equivalence of the respective concrete and abstract semantics.

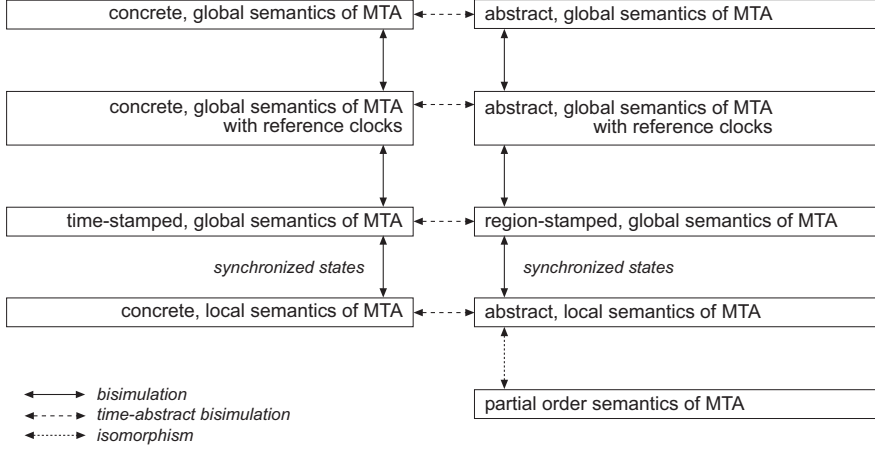


Figure 10: Semantics of MTA and their relations.

Being defined in terms of the underlying transition system, the semantics mentioned above are sequential. An analysis of the abstract, local semantics will yield that concurrently enabled actions are independent. This gives rise to a partial order semantics which we define in terms of a labelled event structure. We prove that the partial order semantics and the sequential semantics define exactly the same set of runs. Section 4 is the dedicated to the proof that the event structure has a complete, finite prefix, which renders its applicable to model-checking techniques.

## 2.1 Basic Definitions

**Definition 1 (Atomic Clock Constraint, Clock Formula).** Let  $C$  be a finite set of *clock variables*. The set  $ACC(C)$  of *atomic clock constraints* is the set of formulas  $x \bowtie k$  and  $x - y \bowtie k$ , for all  $x, y \in C, k \in \mathbb{N}$  where  $\bowtie \in \{\leq, <, =, >, \geq\}$ . A *clock formula* over  $C$  is a finite conjunction of atomic clock constraints over  $C$ .  $CF(C)$  denotes the set of all clock formulas over  $C$ . \*

We consider timed automata with linear ( $x \bowtie k$ ) and diagonal constraints ( $x - y \bowtie k$ ) only since reachability for this class is decidable, whereas allowing additive constraints ( $x + y \bowtie k$ ) in timed automata with more than four clocks renders reachability problems undecidable. [BD00]

## 2.2 Classic Timed Automata

After the informal introduction of *timed automata* in Sect. 1.2 we need to become more precise when talking about a timed automaton's structure and its behaviour. We repeat the definition of a timed automaton (TA), its states and steps, the underlying transition system and the notion of a timed run. At least the latter definition (Def. 6) should be interesting to readers who are familiar with timed automata in order to follow the notation and subsequent definitions.

**Definition 2 (Timed Automaton).** A *timed automaton*  $A = (W, \Gamma, C, \longrightarrow, \text{inv}, w^0)$  consists of

- a finite set  $W$  of *discrete states*,
- a finite *alphabet*  $\Gamma$ ,
- a finite set  $C$  of clock variables,
- the *state invariant function*  $\text{inv} : W \rightarrow CF(C)$ ,
- the *transition relation*  $\longrightarrow \subseteq W \times CF(C) \times \Gamma \times 2^C \times W$ , and
- the designated *initial state*  $w^0$ . \*

Let  $A = (W, \Gamma, C, \longrightarrow, \text{inv}, w^0)$  be fixed for the remainder of this section. A transition  $(w, \varphi, a, R, w') \in \longrightarrow$  is denoted as  $w \xrightarrow{\varphi, a, R} w'$ .

**Definition 3 (Timed State of a Timed Automaton).** Let  $A$  be a timed automaton. A function  $\sigma : C \rightarrow \mathbb{R}_0^+$  is a *clock valuation*; by  $\Sigma(C)$  denote the set of all clock valuations over  $C$ . The pair  $s = (w, \sigma)$ ,  $w \in W$  is a *timed state* of  $A$ .

Let  $S(A)$  denote the set of all timed states of  $A$ . Let  $\sigma^0$  with  $\sigma^0(x) = 0$  for all  $x \in C$ . The state  $s^0 = (w^0, \sigma^0)$  is the *initial* timed state of  $A$ . \*

To simplify the language in the following, we will write ‘state’ instead of ‘timed state’. For a finite set  $C$  of clock variables, let  $\Sigma(C) =_{\text{def}} (C \rightarrow \mathbb{R}_0^+)$  denote the set of all valuations of the clock variables in  $C$ .

Fix an arbitrary finite set  $C$  of clock variables. The interpretation of  $\varphi \in CF(C)$  in a clock valuation  $\sigma \in \Sigma(C)$  is straight forward. We write  $\sigma \models \varphi$  iff  $\sigma$  satisfies  $\varphi$ . For  $R \subseteq C$ , by  $\sigma[R \mapsto 0]$  we denote the function that maps  $x$  to 0 if  $x \in R$  and maps  $x$  to  $\sigma(x)$  otherwise. For any  $d \geq 0$  we denote the clock valuation in which from  $\sigma$  as much as  $d$  time has passed by  $(\sigma + d)(x) =_{\text{def}} \sigma(x) + d$  for all  $x \in C$ .

**Definition 4 (Steps of a Timed Automaton).** Let  $s = (w, \sigma) \in S(A)$  be a state of  $A$  and  $a \in \Gamma$ .

1.  $A$  can perform a *discrete step*  $(w, \sigma) \xrightarrow{a} (w', \sigma')$  if  $q \xrightarrow{\varphi, a, R} q'$ ,  $\sigma \models \varphi$ ,  $\sigma' = \sigma[R \mapsto 0]$ , and  $\sigma' \models \text{inv}(w')$ .
2.  $A$  can perform a *time step*  $(w, \sigma) \xrightarrow{d} (w, \sigma')$  with  $d \geq 0$  if  $\sigma' = \sigma + d$  and for all  $0 \leq d' \leq d$ :  $(\sigma + d') \models \text{inv}(w)$ .
3. By  $s \xrightarrow{d, a} s'$  we denote the *combined step*  $s \xrightarrow{d} s''$ ,  $s'' \xrightarrow{a} s'$  of  $A$ . \*

We explicitly allow time steps  $s \xrightarrow{0} s$  where no time passes for technical reasons, e.g. every discrete step then can be preceded by a time step and therefore every discrete step has a corresponding combined step.

**Definition 5 (Underlying Transition System of a TA).** The underlying labelled transition system  $TS(A)$  of  $A$  is given by  $TS(A) = (S(A), \mathbb{R}_{\geq 0} \cup \Gamma, T_{\mathbb{R}} \cup T_{\Gamma}, s^0)$  with

- states  $S(A)$ ,
- labels  $\mathbb{R}_{\geq 0}$  and  $\Gamma$ ,
- transitions  $T_{\mathbb{R}} \cup T_{\Gamma}$ , where  $T_{\mathbb{R}} = \{(s, d, s') \mid d \in \mathbb{R}_0^+, s \xrightarrow{d} s' \text{ where } s, s' \in S(A)\}$  and  $T_{\Gamma} = \{(s, a, s') \mid a \in \Gamma, s \xrightarrow{a} s' \text{ where } s, s' \in S(A)\}$ , and
- the initial state  $s^0$ . \*

Equivalently, the underlying timed transition system can be represented using combined steps only:  $TS^\circ(A) = (S(A), T^\circ, s^0)$  with the labelled transition relation  $T^\circ = \{(s, d, a, s') \mid (s, d, s'') \in T_{\mathbb{R}}, (s'', a, s') \in T_{\Gamma}\}$ . An action  $a$  is *enabled* in a state  $s$  iff  $TS(A)$  has a transition  $(s, a, s')$ , it is *fireable* iff there exists a delay  $d \geq 0$  such that  $TS^\circ(A)$  has a transition  $(s, d, a, s')$ .

With the idea of a transition system always comes a notion of a *run* which is simply any finite or infinite sequence  $s^0 s^1 s^2 \dots$  of states starting at the designated initial state  $s^0$  and where subsequent states are connected by the system's transition relation. For our purposes it is useful to include the transitions' labels in the notion of a run and we get alternating sequences of states and transition labels:  $s^0(d_1, a_1)s^1(d_2, a_2)\dots$  where for all  $i > 0$ ,  $(s^{i-1}, d_i, a_i, s^i) \in TS^\circ(A)$  is a combined transition. We call such a run a *labelled run*.

However, the notion of continuous time allows the idea of an ongoing time axis along which all actions are ordered and leads to the definition of a timed run where each action is *time-stamped* with the moment in time when the action occurred – since the beginning of the run. This idea is not new in the field of timed automata and it will become crucial in our arguments later. For this reason and since such a run does not follow immediately from the definition of a transition system, we think it is useful to give an explicit definition here.

**Definition 6 (Timed Run of a Timed Automaton, Reachable State).** A *timed run* of  $A$  is a finite or infinite sequence  $\rho = s^0(a_1, t_1)s^1(a_2, t_2)\dots$  where for all  $i > 0$ :

- $d_i = t_i - t_{i-1} \geq 0$  with  $t_0 = 0$ , and
- $s^{i-1} \xrightarrow{d_i, a_i} s^i$  is a combined step in  $TS^\circ(A)$ .

We say that  $t_i$  is the *time stamp* of action  $a_i$ . Let  $Run^\circ(A)$  denote the set of all runs of  $A$ . If  $\rho$  is finite ( $\rho = s^0(t_1, a_1)\dots s^n$ ) then  $|\rho| = n$  denotes the *length* of  $\rho$  as the number of steps and  $state^\circ(\rho) = s^n$  denotes the resulting state of firing each action  $a_i$  at the corresponding global time  $t_i$  in the given order.

A state  $s$  is called *reachable* in  $A$  if  $A$  has a finite run  $\rho$  ending in  $s = state^\circ(\rho)$ . \*

One can easily see that the timed runs of Definition 6 and the labelled runs described above correspond bijectively through the difference of time stamps ( $d_i = t_i - t_{i-1}$ ) and sum of preceding delays ( $t_i = \sum_{j=1}^i d_j$ ). Using timed runs, we can now define an action  $a \in \Gamma$  to be *enabled at time stamp*  $t \in \mathbb{R}_0^+$  in a state  $s$  iff there exists a timed run  $\rho$  with  $state^\circ(\rho) = s$  and  $\rho(a, t)$   $s'$  is a timed run of  $A$ . In this case we write  $(a, t) \in enabled(s)$ .

### 2.3 Message Passing Timed Automata

As a model for distributed timed systems, we propose *networks of message passing timed automata*. Intuitively, such a network consists of a set of timed automata which can communicate by exchanging messages through point-to-point message buffers.

Because of the promising results for netcharts, which we briefly presented in Sect. 1.3.3 of the introduction, we define message passing timed automata as “timed netcharts”, i.e. the progress of time and actions derive from timed automata, where we allow communicating actions: each sending action has exactly one corresponding receiving action, each such pair induces a dedicated buffer connecting both, sending and receiving actions (see Sect. 1.3.3).

Let  $\mathcal{P} = \{1, 2, \dots, P\}$  be fixed in the following. We use  $\mathcal{P}$  to denote the set of components of the system. Let  $M$  be a finite set of *message types* being the disjoint union of sets  $M(p, q)$ , with  $p, q \in \mathcal{P}$ ,  $p \neq q$ . Each  $m \in M(p, q)$  induces a dedicated communication channel  $(p, q, m)$  from  $p$  to  $q$  which is used to exclusively transmit messages of type  $m$ . Messages are put into  $(p, q, m)$  by the action  $p!q(m)$  in component  $p$  and are taken out of the channel by action  $q?p(m)$  in component  $q$  that is different from  $p$ .

Let  $\mathcal{A} = \{A_p\}_{p \in \mathcal{P}}$  be a family of timed automata, where each automaton is given by  $A_p = (W_p, \Gamma_p, C_p, \longrightarrow_p, inv_p, w_p^0)$  and the following properties are satisfied.

1. The alphabet of each automaton  $\Gamma_p = \Gamma_p^I \uplus \Gamma_p^! \uplus \Gamma_p^?$  is the disjoint union of *send actions*  $\Gamma_p^! = \{p!q(m) \mid \exists q \in \mathcal{P} : m \in M(p, q)\}$ , *receive actions*  $\Gamma_p^? \cup \{p?q(m) \mid \exists q \in \mathcal{P} : m \in M(q, p)\}$ , and *internal actions*  $\Gamma_p^I$ . Let  $\Gamma_p^M = \Gamma_p^! \cup \Gamma_p^?$  denote the set of communicating actions.
2. The alphabets are disjoint:  $\forall p, q \in \mathcal{P} : p \neq q \Rightarrow \Gamma_p \cap \Gamma_q = \emptyset$ .
3. Each communicating action occurs exactly once, i.e. for all  $p \in \mathcal{P}$  and for all  $a \in \Gamma_p^M$ ,  $\longrightarrow_p$  contains exactly one transition labelled with  $a$ .
4. The sets of clock variables are disjoint:  $\forall p, q \in \mathcal{P} : p \neq q \Rightarrow C_p \cap C_q = \emptyset$ .

For every component  $p$  and each of its actions  $a \in \Gamma_p$  define the *location* of  $a$  to be  $loc(a) =_{def} p$ . For each clock variable  $x \in C_p$  of  $p$ , its *location* is defined similarly by  $loc(x) =_{def} p$ .

Observe that by the definition of  $M$  and the  $\Gamma_p^M$  communication is structurally well-defined, i.e. for each sending action  $p!q(m)$  in  $p$  exists a receiving action  $q?p(m)$  in  $q$  and vice versa.

It is reasonable to consider a family  $\mathcal{A}$  of TA only if  $\mathcal{A}$  does not consist of isolated components. We call  $\mathcal{A}$  a *network of message passing timed automata* (MTA) if the

components are connected by communicating actions, i.e. the undirected graph with nodes  $\mathcal{P}$  and edges  $\{\{p, q\} \mid p, q \in \mathcal{P}, p \neq q, M(p, q) \neq \emptyset\}$  is connected.  $\mathcal{A}$  can be represented as a netchart where states and transitions are annotated with invariants and guards from  $CF(C_p)$ , for each  $p \in \mathcal{P}$  respectively.

**Definition 7 (Timed State of  $\mathcal{A}$ ).** A *timed state*  $s = ((w_1, \dots, w_P), \sigma, \beta)$  of a MTA  $\mathcal{A}$  consists of a vector of discrete states ( $w_p \in W_p$  for all  $p \in \mathcal{P}$ ), a clock valuation of all clocks of all components ( $\sigma \in \Sigma(\bigcup_{p \in \mathcal{P}} C_p)$ ), and the *buffer state function*  $\beta : \mathcal{P} \times \mathcal{P} \times M \rightarrow \mathbb{N}$ , where  $\beta(p, q, m)$  denotes the number of messages in channel  $(p, q, m)$ . Let  $S(\mathcal{A})$  be the set of all timed states of  $\mathcal{A}$ .

We write  $\mathbf{w}^0$  for  $(w_1^0, \dots, w_P^0)$ . By  $\beta^0$  we denote the *empty* buffer state function:  $\forall p, q \in \mathcal{P} \forall m \in M : \beta^0(p, q, m) = 0$ . The *initial* timed state of  $\mathcal{A}$  is  $s^0 = (\mathbf{w}^0, \sigma^0, \beta^0)$ .  $\star$

In order to model the behaviour of an MTA  $\mathcal{A}$  we need means to express the change of the contents of buffers due to send and receive actions. A send action  $p!q(m)$  changes the state of the channel  $(p, q, m)$  by adding an undistinguishable message to it. A receive action  $p?q(m)$  removes one message from the channel and hence is enabled only, if the channel is not empty. Having chosen to express the contents of the system's buffers by a buffer state function, we now define the operator  $succ_{buff}$  to transform one buffer state function  $\beta$  into its successor under a given action  $a$ :  $succ_{buff}(a, \beta) = \beta'$  where

$$\beta'(p, q, m) =_{def} \begin{cases} \beta(p, q, m) + 1, & \text{if } a = p!q(m) \in \Gamma_p^M \\ \beta(p, q, m) - 1, & \text{if } a = q?p(m) \in \Gamma_q^M \\ \beta(p, q, m), & \text{otherwise} \end{cases}$$

Note that  $succ_{buff}$  could also yield negative buffer state values if it was applied in an ill-defined way. The definition of a step of an MTA given in the following will prevent such an application.

Like a timed automaton, a MTA has discrete steps and time steps. A discrete step is performed by exactly one component of the network while all other components stay put; time does not pass in a discrete step. Contrarily, in a time step all components advance in time simultaneously by the same amount of time.

By  $\xrightarrow{d}$ ,  $\xrightarrow{a}$ ,  $\xrightarrow{d,a}$  we denote time steps, discrete steps, and composed steps of  $A_p$ , respectively. By  $\sigma|_p$  we denote the restriction of  $\sigma$  to clocks from  $C_p$ .

**Definition 8 (Step of  $\mathcal{A}$ ).** Let  $s = ((w_1, \dots, w_P), \sigma, \beta) \in S(\mathcal{A})$  be a state.

1. The state  $s' = ((w'_1, \dots, w'_P), \sigma', \beta') \in S(\mathcal{A})$  can be reached from  $s$  by a *discrete step*  $s \xrightarrow{a} s'$  iff there exists  $p \in \mathcal{P}$  such that
  - a)  $a \in \Gamma_p$  ( $a$  is an action of component  $p \in \mathcal{P}$ ),
  - b)  $(w_p, \sigma|_p) \xrightarrow{a} (w'_p, \sigma'|_p)$  ( $A_p$  has a corresponding discrete step),
  - c)  $a = p?q(m) \in \Gamma_p^? \Rightarrow \beta(q, p, m) > 0$  (if  $a$  is a receive action, then the corresponding channel contains a message),
  - d)  $\beta' = succ_{buff}(a, \beta)$ , and

- e) for all  $q \in \mathcal{P}$ ,  $p \neq q$  implies  $w'_q = w_q$  and  $\sigma'|_q = \sigma|_q$  (all other components do not change their state).
2. The state  $s' = ((w'_1, \dots, w'_P), \sigma', \beta) \in S(\mathcal{A})$  can be reached from  $s$  by a *time step*  $s \xrightarrow{d} s'$ ,  $d \geq 0$  iff all components of  $\mathcal{A}$  can make a time step of length  $d$ : for all  $p \in \mathcal{P}$ ,  $(w_p, \sigma|_p) \xrightarrow{d}_p (w'_p, \sigma'|_p)$ , and  $\beta' = \beta$ .
3. The state  $s' \in S(\mathcal{A})$  can be reached by a *composed step*  $s \xrightarrow{d,a} s'$  iff there exists  $s'' \in S(\mathcal{A})$  such that  $s \xrightarrow{d} s''$  and  $s'' \xrightarrow{a} s'$ . \*

The definition of the underlying transition systems  $TS(\mathcal{A})$ ,  $TS^\circ(\mathcal{A})$ , and a timed run of  $\mathcal{A}$ , enabledness in a state, and reachability in  $\mathcal{A}$  are straight-forward by Definitions 5 and 6 given for timed automata above.

A MTA is *B-bounded* if there exists  $B \in \text{natnum}$  such that in every reachable state  $s$  of  $TS(\mathcal{A})$  the number of messages in every channel is at most  $B$ , i.e. with  $s = (\mathbf{w}, \sigma, \beta)$ ,  $\forall p, q \in \mathcal{P} \forall m \in M(p, q) : \beta(p, q, m) \leq B$ .

## 2.4 Global and Local Time Steps

The semantics of an MTA as it is given in Sect. 2.3 already has a notion of locality in its discrete steps where only one component is allowed to change the state and the parts of the state that may be affected by a discrete step are limited: in a discrete step  $s = ((w_1, \dots, w_P), \sigma, \beta) \xrightarrow{a} ((w'_1, \dots, w'_P), \sigma', \beta')$ , action  $a$  with location  $p = \text{loc}(a)$  can change at most the discrete state of  $p$ , the valuation of the clock variables of  $p$  and the contents of the buffers going to and from  $p$ :

$$\begin{aligned} \forall q \in \mathcal{P} : p \neq q \Rightarrow w_q = w'_q \wedge \sigma|_q = \sigma'|_q \\ \wedge \forall r \in \mathcal{P} : r \neq q \Rightarrow \forall m \in M(q, r) : \beta(q, r, m) = \beta'(q, r, m). \end{aligned}$$

But by definition, a time-step affects the clock valuation of all components and hence cannot be considered local. Bengtsson et al. observed this fact and suggested to localize time-steps by letting time pass in a single component only while all other components stay put [BJLY98]. The original system semantics as we defined it in Sect. 2.3 requires time to pass in all components by the same amount. The semantics in the next section introduces timely de-synchronized states where different amount of time has passed in different parts of the system since a run began. This allows actions to occur in a timely unordered way if they are not ordered causally.

It is necessary to keep in mind that only those states where the same amount of time has totally passed in every component may correspond to states that are reachable in the global, concrete semantics.

## 2.5 Reference Clocks and Time Stamping Messages

These observations of Sect. 2.4 lead to the insight that it is necessary to introduce additional book keeping on how much time has passed in total in each component. Following

the approach of Bengtsson et al. [BJLY98] we will introduce a reference clock variable for each component which is never reset and thus evaluates in every state to the amount of time that has passed until that state has been reached.

Let  $A = (W, \Gamma, C, \longrightarrow, \text{inv}, w^0)$  be a timed automaton. A clock variable  $g \in C$  is a *reference clock variable* if for all  $q \xrightarrow{\varphi, a, R} q'$ ,  $g \notin R$  and  $\varphi \in CF(C \setminus \{g\})$ , and for all  $w \in W$ :  $\text{inv}(w) \in CF(C \setminus \{g\})$ , i.e.  $g$  does not occur in any guard or invariant of  $A$  and  $g$  is never reset. To simplify the notation and since a reference clock variable is never written, we will use the term *reference clock* instead.

**Adding Reference Clocks to Timed Automata** Let  $A$  be a timed automaton without a reference clock. Let  $g$  be a clock variable that is not in  $C$ . We call the timed automaton  $A + g =_{\text{def}} (W, \Gamma, C \uplus \{g\}, \longrightarrow, \text{inv}, w^0)$  a TA *with reference clock*  $g$ . For  $C \uplus \{g\}$  we will write  $C^+$ . Let  $A$  and  $g$  be fixed for the remainder of this section.

As the reference clock of  $A + g$  is never reset in any timed run of the system – and by definition evaluates to 0 in the initial timed state of  $A + g$  –, its value in a state is the sum of all delays that was needed to reach the state. This observation leads to the following proposition.

**Proposition 2:** *Let  $\rho = s^0(a_1, t_1)s^1(a_2, t_2) \dots$  be a timed run of a timed automaton  $A + g$  with reference clock  $g$ . Then for all  $1 \leq i \leq |\rho|$  and states  $s^i = (\mathbf{w}^i, \sigma^i, \beta^i)$  holds:  $\sigma^i(g) = t_i$ .* \*

*Proof.* The proposition holds because in each time-step all clocks, including the reference clocks, are increased by the same amount  $d \in \mathbb{R}_{\geq 0}$ . Then by the observation of Sect. 2.2, that the sum of the preceding delays  $d_j$  yields the the time-stamp  $t_i$  of an action, the proposition holds.  $\square$

From Proposition 2 we see that the introduction of the reference clock  $g$  actually renders time stamps of  $A$ 's actions a part of its transition system. In a run of a timed automaton with reference clock, an action's *time stamp* is the value of the reference clock after that action occurred. This relation will go into a re-definition of timed runs of networks of timed automata later in this section.

Since  $g$  is does not occur in any invariant or guard of  $A$ , it cannot affect the behaviour of  $A + g$  and we see that the following proposition holds.

**Proposition 3:** *Let  $A$  be a timed automaton without reference clocks and let  $g$  be the reference clock in  $A + g$ . Then  $A + g$  and  $A$  are bisimilar.* \*

*Proof.* For the bisimilarity relation that relates a state of  $A$  to a state of  $A + g$  iff both states coincide on the discrete states, on the clock valuations of all clocks except  $g$ , the proposition follows.  $\square$

**Adding Reference Clocks to MTA** Let  $\mathcal{A}$  be a network of message-passing timed automata. Let  $g_1, \dots, g_P \in \text{Clocks} \setminus (\bigcup_p C_p)$  be pairwise distinct. By  $\mathcal{A}^+$  we denote  $\{A_p + g_p\}_{p \in \mathcal{P}}$  the network of message-passing timed automata *with local reference clocks*  $\mathbf{g} = (g_1, \dots, g_P)$ , abbreviated  $\text{MTA}^+$ . Additionally, define  $C^+ =_{\text{def}} \bigcup_{p \in \mathcal{P}} C_p^+ = \bigcup_{p \in \mathcal{P}} C_p \uplus \{g_p\}$ .

Since an  $\text{MTA}^+$  is an “ordinary” MTA, the semantics of Sect. 2.3 are applicable and in any thereby reachable state the values of all reference clocks are equal. This redundancy of information does no harm in the current definition of  $\text{MTA}^+$  and it will facilitate proofs later on when we examine semantics where reference clocks may be no longer equal.

As above for timed automata, the time stamp of an action in  $\mathcal{A}^+$  is the value of its component’s reference clock after the action occurred.

For the purpose of correctness proofs, we will time stamp messages from now on. That is, a message *is* the time stamp of the sending action in the moment of sending. And we requires that, although this is trivially satisfied in  $TS(\mathcal{A}^+)$ , a message can be received only, if its time stamp is not later than the time stamp of the receiving action upon execution.

**Definition 9 (Time-Stamped State of Networks of Timed Automata).** Let  $\mathcal{A}^+$  be a  $\text{MTA}^+$ . A function  $\beta : \mathcal{P} \times \mathcal{P} \times M \rightarrow \mathcal{M}(\mathbb{R}_0^+)$  is a *time-stamped buffer state function*, where  $\beta(p, q, m)$  contains the time-stamps of the messages in channel  $(p, q, m)$ . Hence  $\beta^0$  with  $\beta^0(p, q, m) = []$  for all  $p, q \in \mathcal{P}$ ,  $m \in M$  is the *empty* buffer state function.

With discrete state  $\mathbf{w} \in W_1 \times \dots \times W_P$ , clock assignment  $\sigma \in \Sigma(\bigcup_{p \in \mathcal{P}} C_p^+)$ , and  $\beta$  a time-stamped buffer state function,  $(\mathbf{w}, \sigma, \beta)$  is a *time-stamped state* of  $\mathcal{A}^+$ . As usual,  $(\mathbf{w}^0, \sigma^0, \beta^0)$  is its initial state. The set of states of  $\mathcal{A}^+$  is  $S^{\mathbf{g}}(\mathcal{A}^+)$ . \*

Changing the way of representing the state of the buffers, we need to adapt the definition of the buffer state successor operator  $\text{succ}_{\text{buff}}$ :  $\text{succ}_{\text{buff}}(a, t, \beta)$  is the time-stamped buffer state function that is created by firing action  $a$  with time-stamp  $t$  in the buffer state  $\beta$ :  $\text{succ}_{\text{buff}}(a, t, \beta) = \beta'$  where

$$\beta'(p, q, m) =_{\text{def}} \begin{cases} \beta(p, q, m) + t, & \text{if } a = p!q(m) \in \Gamma_p^M \\ \beta(p, q, m) - t', & \text{if } a = q?p(m) \in \Gamma_q^M, \\ & t' \in \beta(p, q, m), t' \leq t \\ \beta(p, q, m), & \text{otherwise} \end{cases}$$

Having all pieces together, we may now define a step of an  $\text{MTA}^+$  where messages are time-stamped. The actual difference to the notion of a step for MTA as we have given it in Definition 8 is the treatment of the buffer-state function. Therefore the values of reference clocks in different components will still be equal in the corresponding runs.

**Definition 10 (Time-Stamped Semantics of Networks of Timed Automata).** Let  $\mathcal{A}^+$  be an  $\text{MTA}^+$  with reference clocks  $\mathbf{g} = (g_1, \dots, g_P)$ . Let  $s$  be a time-stamped state,  $s = ((w_1, \dots, w_P), \sigma, \beta) \in S^{\mathbf{g}}(\mathcal{A}^+)$ .

1. The state  $s' = ((w'_1, \dots, w'_p), \sigma', \beta') \in S^{\mathbf{g}}(\mathcal{A}^+)$  can be reached from  $s$  by a *discrete step*  $s \xrightarrow{a} s'$  iff there exists  $p \in \mathcal{P}$  such that
  - a)  $a \in \Gamma_p$ ,
  - b)  $(w_p, \sigma|_p) \xrightarrow{a} (w'_p, \sigma'|_p)$ ,
  - c) if  $a = p?q(m) \in \Gamma_p^?$  then  $\exists t' \in \beta(q, p, m) : t' \leq \sigma(g_p)$  holds,
  - d)  $\beta' = \text{succ}_{\text{buff}}(a, \sigma(g_p), \beta)$ , and
  - e) for all  $q \in \mathcal{P}$ ,  $p \neq q$  implies  $w'_q = w_q$ ,  $\sigma'|_q = \sigma|_q$
2. The state  $s' = ((w'_1, \dots, w'_p), \sigma', \beta') \in S^{\mathbf{g}}(\mathcal{A}^+)$  can be reached from  $s$  by a *time step*  $s \xrightarrow{d} s'$ ,  $d \geq 0$  iff for all  $p \in \mathcal{P}$ ,  $(w_p, \sigma|_p) \xrightarrow{d} (w'_p, \sigma'|_p)$ , and  $\beta' = \beta$ .
3. The state  $s' \in S(\mathcal{A}^+)$  can be reached by a *composed step*  $s \xrightarrow{d,a} s'$  iff there exists  $s'' \in S(\mathcal{A}^+)$  such that  $s \xrightarrow{d} s''$  and  $s'' \xrightarrow{a} s'$ . \*

The definition of the underlying *time-stamped transition system* of  $\mathcal{A}^+$  a  $\text{MTA}^+$  with reference clocks  $\mathbf{g}$  results directly from the notion of step as given above (see Definition 5). By  $TS^{\mathbf{g}}(\mathcal{A}^+)$  denote the transition system constituted by discrete and time steps, by  $TS^{\circ, \mathbf{g}}(\mathcal{A}^+)$  denote the system constituted by composed steps.

Unlike for classic timed automata or  $\text{MTA}$  without reference clocks, a timed run now follows directly from the definition of  $TS^{\circ, \mathbf{g}}(\mathcal{A}^+)$  since time-stamps have become part of the state: a *timed run* is a way in the transition system where the transition between two states is annotated with its action's time stamp; we modify the definition of a timed run accordingly and introduce the notion of a *time-stamped action sequence*.

**Definition 11 (Time-Stamped Run of a  $\text{MTA}$ ).** Let  $\mathcal{A}^+$  be a  $\text{MTA}^+$  with reference clocks  $\mathbf{g} = (g_1, \dots, g_P)$ . A *timed run* of  $\mathcal{A}^+$  is an alternating sequence of states  $s_i \in S(\mathcal{A}^+)$  and time-stamped actions  $(a_i, t_i) \in \Gamma \times \mathbb{R}_0^+$ ,  $\rho = s^0(a_1, t_1)s^1(a_2, t_2) \dots$  where for all  $i > 0$ :

- $s^i = (\mathbf{w}^i, \sigma^i, \beta^i)$ ,  $t_i = \sigma^i(g_p)$  with  $p = \text{loc}(a_i)$ ,
- $d_i = t_i - t_{i-1}$  with  $t_0 := 0$ , and
- $(s^{i-1}, d_i, a_i, s^i)$  is a transition in  $TS^{\circ, \mathbf{g}}(\mathcal{A}^+)$ .

Let  $\text{Run}^{\circ, \mathbf{g}}(\mathcal{A}^+)$  denote the set of all timed runs of  $\mathcal{A}^+$  and let  $\text{state}^{\circ, \mathbf{g}}(\rho)$  denote the last state of a finite run  $\rho \in \text{Run}^{\circ, \mathbf{g}}(\mathcal{A}^+)$ .

By  $\rho_S = s_0 s_1 s_2 \dots$  and  $\hat{\rho} = (a_1, t_1)(a_2, t_2) \dots$  we denote the subsequences of states and time-stamped actions of  $\rho$ , respectively. We will call  $\hat{\rho}$  a *time-stamped action sequence* (ta-sequence for short) and write  $\text{TaSeq}^{\circ, \mathbf{g}}(\mathcal{A}^+)$  for the set of all ta-sequences of  $\mathcal{A}^+$ . \*

The notion of the length  $|\hat{\rho}|$  of a finite ta-sequence  $\hat{\rho}$  and the resulting  $\text{state}^{\circ, \mathbf{g}}(\hat{\rho})$  carry over from runs. Similarly, the notion of reachable states and enabled actions in a state are straight-forward from those in Sect. 2.2.

Stressing again the observation that for all  $p, q \in \mathcal{P}$  and for all reachable states  $(\mathbf{w}, \sigma, \beta)$  (from  $s^0$ ),  $\sigma(g_p) = \sigma(g_q)$  holds, we can conclude an important result. Any message that is sent in the time-stamped semantics can be received iff it can be received in the standard semantics.

**Proposition 4:** *Let  $\mathcal{A}$  be an MTA and  $\mathcal{A}^+$  the corresponding  $MTA^+$ . The transition systems  $TS(\mathcal{A})$  and  $TS^g(\mathcal{A}^+)$  are bisimilar.* \*

*Proof.* The bisimilarity relation between  $\mathcal{A}$  and  $\mathcal{A}^+$  relates states that coincide on the discrete states, on the clock valuations for all but the reference clocks and where the same number of messages is in every buffer. The equality of the number of messages is sufficient because in reachable states, time-stamps of messages are always less than or equal to the reference clocks of the receiver. □

## 2.6 Local Clock Semantics

We will now allow different components to advance in their time independently of the others. A time step of  $\mathcal{A}^+$  no longer requires all components to advance in time by the same amount of time.

To emphasize this in notation, we decompose the state into the parts of its components: each component  $p$  has its own discrete state  $w_p \in W_p$  and its own clock valuation  $\sigma_p \in \Sigma(C_p^+)$ , the buffer state function  $\beta$  remains in its current form since it describes a part of the system that lies between different components. Let  $s = ((w_1, \dots, w_P), \sigma, \beta) \in S(\mathcal{A}^+)$  be a time-stamped state. By  $\pi_{\mathcal{P}}(s)$  we denote the state  $((w_1, \sigma_1), \dots, (w_P, \sigma_P), \beta)$  where  $\forall p \in \mathcal{P} : \sigma_p = \sigma|_p$ .

**Lemma 5:**  *$\pi_{\mathcal{P}}$  is a bijection.*

*Proof.* Follows trivially, because the sets of clock variables of any two components are pairwise disjoint. □

We write  $\pi_{\mathcal{P}}^{-1}$  for the inverse of  $\pi_{\mathcal{P}}$ , i.e. the composition of the states of all components into a joint state. Accordingly, for a set  $S$  of states by  $\pi_{\mathcal{P}}(S) =_{def} \{\pi_{\mathcal{P}}(s) \mid s \in S\}$ , the set of decomposed states of  $\mathcal{A}^+$  is  $S_{\mathcal{P}}^g(\mathcal{A}^+) =_{def} \pi_{\mathcal{P}}(S(\mathcal{A}^+))$ .

With the new notation at hand, we can now give a definition of a *local discrete step* and – more importantly – of a *local time step*, that modify the discrete mode and the clock valuation of a single component only.

**Definition 12 (Local Clock Semantics of Networks of Timed Automata).** Let  $s = ((w_1, \sigma_1), \dots, (w_P, \sigma_P), \beta) \in S_{\mathcal{P}}^g(\mathcal{A}^+)$  be a state.

1. The state  $s' = ((w'_1, \sigma'_1), \dots, (w'_P, \sigma'_P), \beta')$  can be reached by a *local discrete step*  $s \xrightarrow{a}_p s'$  of component  $p \in \mathcal{P}$  iff the conditions of Definition 10.(1) hold for a discrete step  $\pi_{\mathcal{P}}^{-1}(s) \xrightarrow{a}_p \pi_{\mathcal{P}}^{-1}(s')$ ,  $a \in \Gamma_p$ .

2. The state  $s' = ((w'_1, \sigma'_1), \dots, (w'_P, \sigma'_P)), \beta'$  can be reached by a *local time step*  $s \xrightarrow{d}_p s'$ ,  $d \geq 0$  of component  $p \in \mathcal{P}$  iff  $(w_p, \sigma_p) \xrightarrow{d}_p (w'_p, \sigma'_p)$  and for all  $q \in \mathcal{P}$ ,  $p \neq q \Rightarrow w_q = w'_q \wedge \sigma_q = \sigma'_q$ , and  $\beta' = \beta$ .
3. The state  $s' \in S_l^{\mathbf{g}}(\mathcal{A}^+)$  can be reached by a *composed step*  $s \xrightarrow{d,a} s'$  iff there exists  $p \in \mathcal{P}$  and  $s'' \in S_l^{\mathbf{g}}(\mathcal{A}^+)$  such that  $s \xrightarrow{d}_p s''$  and  $s'' \xrightarrow{a}_p s'$ . \*

Observe that a local discrete step of component  $p$  still can modify the discrete state and the clock valuation of  $p$  and the contents of buffers from and to  $p$  only. Additionally, a local time step of component  $p$  modifies the clock valuation of  $p$  only.

The underlying transition systems  $TS_l^{\mathbf{g}}(\mathcal{A}^+)$  and  $TS_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$  follows from the definition of a local step as given above. Since  $\mathcal{A}^+$  has a reference clock for each component, the transition system  $TS_l^{\mathbf{g}}(\mathcal{A}^+)$  is time-stamped. Thus the notion of a *timed run* ( $Run_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$ ), of a *ta-sequence* ( $TaSeq_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$ ), of *reachable* states ( $state_l^{\circ, \mathbf{g}}(\rho)$ ), and of *enabled* (time-stamped) actions follows the same lines as in Definition 11.

By allowing time to pass locally in a component, we introduce new actions that lead to newly reachable states. For the same MTA<sup>+</sup>  $\mathcal{A}^+$ ,  $TS_l^{\mathbf{g}}(\mathcal{A}^+)$  allows more behaviour than  $TS^{\mathbf{g}}(\mathcal{A}^+)$ . Through time-stamping messages we disallow non-causal behaviour: a message can be received only if the receiver's time-stamp is not earlier than the message's time-stamp which is the time-stamp of its sender in the moment of sending. This excludes non-causal behaviour of “receiving before sending”. In order to see how states and runs in  $TS^{\mathbf{g}}(\mathcal{A}^+)$  relate to those in  $TS_l^{\mathbf{g}}(\mathcal{A}^+)$ , we need a new term.

**Definition 13 (Synchronized and Synchronizable States).**

Let  $s = ((w_1, \sigma_1), \dots, (w_P, \sigma_P), \beta) \in S_l^{\mathbf{g}}(\mathcal{A}^+)$  be a state.  $s$  is called *synchronized* iff all reference clocks are equal:  $\forall p, q \in \mathcal{P} : \sigma_p(g_p) = \sigma_q(g_q)$ .  $s$  is called *synchronizable* iff there exist delays  $d_1, \dots, d_P \in \mathbb{R}_0^+$  and states  $s_1, \dots, s_P \in S_l^{\mathbf{g}}(\mathcal{A}^+)$  such that

$$s \xrightarrow{d_1}_1 s_1 \xrightarrow{d_2}_2 \dots s_{P-1} \xrightarrow{d_P}_P s_P$$

and  $s_P$  is synchronized. \*

In synchronized states, the repeatedly observed property of equal reference clocks is satisfied; a state  $s$  that is reachable by time-stamped semantics is synchronized. In fact, this property is necessary and sufficient in local clock semantics to represent a state that is reachable in time-stamped semantics.

**Theorem 6.** *For every state  $s \in S(\mathcal{A}^+)$ ,  $s$  can be reached from  $s^0$  by time-stamped semantics iff  $\pi_{\mathcal{P}}(s)$  can be reached from  $\pi_{\mathcal{P}}(s^0)$  by local clock semantics.* \*

Our definitions are in compliance with the definitions of Minea. Therefore the proof of this result follows the corresponding proof in [Min99b] and is omitted here.

## 2.7 Region Abstraction of Timed Automata

In the last sections, we considered the state space of timed automata where the value of a clock variable in a state is a real number. This state space is infinite and not verifiable through model-checking. The crucial achievement of Alur and Dill is to have shown that it can finitely be represented. The necessary restriction on timed automata for this result is that guards and invariants consist of clock constraints with rational numbers only. These can be transformed into constraints on natural numbers (by multiplication with the least common denominator), which “stretches” time only without affecting the reachability of states. [AD94]

Intuitively, such a timed automaton “slices” the time into finitely many pieces, where in each piece, guards and invariants yield the same truth value for different clock valuations. Regarding clocks that may grow arbitrarily large: a timed automaton  $A$  is finite and hence there is a largest natural number  $k_x$  occurring in  $A$  against which the clock variable  $x$  is compared. If  $x$  is larger than  $k_x$ , clock constraints over  $x$  cannot change their truth value by a progress of time. Alur and Dill identified the *region equivalence* of clock valuations as well as the *zone abstraction* that turn this observation into a formal representation [AD94].

In this thesis we will apply the region equivalence to finitely represent the clock valuations of the system. Although zones, being convex unions of regions, have a better average case complexity than regions, we stick to the latter to avoid conceptual problems in the abstraction. We provide a proper reason for our decision in Sect. 2.8.1 when introducing abstract time-stamps for the local abstract semantics.

Let  $A = (W, \Gamma, C, \longrightarrow, inv, w^0)$  be a timed automaton. For each  $x \in C$  let  $k_x$  be largest integer  $k$  such that  $(x \leq c)$  or  $(c \geq x)$  is a clock constraint in  $A$ .<sup>6</sup> The *region-equivalence relation*  $\sim_r$  is defined over all clock valuations of  $A$ ;  $\sigma \sim_r \sigma'$  iff all of the following conditions hold:

1. For all  $x \in C$ , either  $\lfloor \sigma(x) \rfloor = \lfloor \sigma'(x) \rfloor$  or  $\sigma(x) > k_x$  and  $\sigma'(x) > k_x$ , where  $\lfloor r \rfloor$  denotes the integral part of  $r \in \mathbb{R}_{\geq 0}$ .
2. For all  $x, y \in C$  with  $\lfloor \sigma(x) \rfloor < k_x$  and  $\lfloor \sigma(y) \rfloor < k_y$  holds:  $fract(\sigma(x)) \leq fract(\sigma(y))$  iff  $fract(\sigma'(x)) \leq fract(\sigma'(y))$ , where  $fract(r) = r - \lfloor r \rfloor$  denotes the fractional part of  $r \in \mathbb{R}_{\geq 0}$ .
3. For all  $x \in C$  with  $\lfloor \sigma(x) \rfloor < k_x$  holds:  $fract(\sigma(x)) = 0$  iff  $fract(\sigma'(x)) = 0$ .

By  $[\sigma]_r$  denote the equivalence class of  $\sigma$  in  $\sim_r$ , a *clock region* over  $C$ . We usually use the symbol  $\alpha$  to denote a clock region, and we write  $\sigma \in \alpha$  if  $\sigma$  is a member of the equivalence class  $\alpha$ , as usual. Let  $R(C)$  be the set of all clock regions over  $C$ .

Alur and Dill have shown that the number of regions is finite, and that region equivalent clock valuations satisfy the same set of clock constraints occurring in  $A$ , i.e. all clock

---

<sup>6</sup>We restricted clock constraints from the very beginning to natural numbers, see Def. 1.

valuations in a clock region equivalently evaluate the guards and invariants of  $A$ . Therefore, satisfaction of clock constraints extends to clock regions canonically. Furthermore every clock region can be represented by a set of clock constraints:

1. for every clock  $x \in C$  exactly one constraint from

$$\{x = k \mid k = 0, 1, \dots, k_x\} \cup \{k < x < k + 1 \mid k = 0, 1, \dots, k_x - 1\} \cup \{x > k_x\}$$

2. and for all clocks  $x, y \in C, x \neq y$  with the constraints  $k < x < k + 1$  and  $k' < y < k' + 1$  from 1. for some  $k, k' \in \mathbb{N}$  one constraint from

$$\{\text{fract}(x) < \text{fract}(y), \text{fract}(x) = \text{fract}(y), \text{fract}(x) > \text{fract}(y)\}.$$

Additionally, for equivalent clock valuations  $\sigma$  and  $\sigma'$  holds that for every  $d \in \mathbb{R}_{\geq 0}$  exists a  $d' \in \mathbb{R}_{\geq 0}$  such that  $\sigma + d \sim_r \sigma' + d'$ .

This justifies the notion of the *successor region* of  $\alpha$  being the region  $\text{succ}_r(\alpha)$  that is reached by letting time pass in  $\alpha$ : if in  $\alpha, x > k_x$  holds for all  $x \in C$  then  $\text{succ}_r(\alpha) = \alpha$ , otherwise  $\text{succ}_r(\alpha)$  is the unique clock region  $\alpha' \neq \alpha$  for which for any  $\sigma \in \alpha$  exists some  $d > 0$  such that  $\sigma + d \in \alpha'$  and for all  $0 < c < d, \sigma + c \in \alpha \cup \alpha'$ . The set  $\text{succ}_r^*(\alpha)$  is the smallest set containing  $\alpha$  such that for all  $\alpha' \in \text{succ}_r^*(\alpha), \text{succ}_r(\alpha') \in \text{succ}_r^*(\alpha)$  holds. Additionally we need the set of all *strict* successor regions  $\text{succ}_r^+(\alpha) = \text{succ}_r^*(\alpha) \setminus \{\alpha\}$ .

Resetting clocks preserves region equivalence as well: for  $R \subseteq C, \sigma[R \mapsto 0] \sim_r \sigma'[R \mapsto 0]$ , hence the reset-operation extends to regions canonically:  $\alpha[R \mapsto 0] =_{\text{def}} \{\sigma[R \mapsto 0] \mid \sigma \in \alpha\}$ .<sup>7</sup>

By  $R(A)$  we denote the *region automaton* induced by  $\sim_r$  on  $TS(A)$ . States of  $R(A)$  are tuples  $(w, \alpha), w \in W, \alpha \in R(C)$ , called *regions*, the set of all regions of  $R(A)$  is  $RS(A)$ . The initial state is  $(w^0, [\sigma^0]_r)$ .

For regions  $(w, \alpha), (w', \alpha')$  we write

- $(w, \alpha) \xrightarrow{a} (w', \alpha')$  to denote the *discrete step* in  $R(A)$  due to action  $a$  ( $w \xrightarrow{\varphi, a, R} w'$ ), where  $\alpha \models \varphi, \alpha' = \alpha[R \mapsto 0], \alpha' \models \text{inv}(w')$ , and
- $(w, \alpha) \xrightarrow{\delta} (w', \alpha')$  to denote a *time step* in  $R(A)$ , i.e. if  $w = w'$  and  $\alpha' \in \text{succ}_r^+(\alpha)$  and  $\alpha' \models \text{inv}(w)$ . Please note that the label  $\delta$  is a mere symbol and not a parameter like the delay  $d$  in concrete time steps of  $A$ 's transition system.

## 2.8 Timed-Stamped Region Semantics of Networks of Message Passing Timed Automata

Having defined  $\text{MTA}^+$  as a specific composition of timed automata, a  $\text{MTA}^+ \mathcal{A}^+$ 's transition system  $TS(\mathcal{A}^+)$  is infinite but gives rise to a finite representation through region equivalence. Our ultimate goal is to obtain a finite representation for the transition system  $TS_l^g(\mathcal{A}^+)$  due to local clock semantics as it is defined in Sect. 2.6. We have chosen

<sup>7</sup>Both,  $\text{succ}_r(\cdot)$  and the reset-operation can symbolically be defined on the sets of clock constraints that specify a region.

to use the region abstraction to achieve this aim; we will therefore examine the region abstraction of a MTA in standard semantics first, followed by the region abstraction of a MTA's time-stamped transition system to finally desynchronize the progress of time.

A network of message passing timed automata  $\mathcal{A}$  (without a reference clock) has a straightforward underlying *region automaton*  $R(\mathcal{A})$  with regions  $((w_1, \dots, w_P), \alpha, \beta) \in RS(\mathcal{A})$ ,  $\forall p \in \mathcal{P} : w_p \in W_p$ ,  $\alpha \in R(C)$ , and  $\beta$  a buffer-state function as in Definition 3. Discrete steps modify the region as given above and alter the buffer-state function as in Definition 4; time steps are as given above. The set of runs and ta-sequences of  $R(\mathcal{A})$  are  $Run_R^\circ(\mathcal{A})$  and  $TaSeq_R^\circ(\mathcal{A})$ , respectively.

**Proposition 7:** *Let  $\mathcal{A}$  be a network of message passing timed automata without reference clocks. A region  $(\mathbf{w}, \alpha, \beta) \in RS(\mathcal{A})$  is reachable in  $R(\mathcal{A})$  iff there exists  $\sigma \in \alpha$  such that  $(\mathbf{w}, \sigma, \beta)$  is reachable in  $TS(\mathcal{A})$ . Furthermore, if  $\mathcal{A}$  is  $B$ -bounded, then the set of reachable states in  $R(\mathcal{A})$  is finite.* \*

*Proof.* Proof Sketch: Assume the semantics where receive actions  $q?p(m)$  can occur regardless of the presence of a message (i.e. every action can occur if its guard is satisfied). Let  $TS'(\mathcal{A})$  represent accordingly the underlying transition system of the parallel composition of  $A_p, p \in \mathcal{P}$  and  $R'(\mathcal{A})$  the corresponding region automaton which is a correct abstraction according to the region equivalence (see proof of correctness of region abstraction in [AD94]).

Let  $s = (\mathbf{w}, \sigma, \beta)$ ,  $s_R = (\mathbf{w}, \alpha, \beta_R)$ ,  $\sigma \in \alpha$  be reachable in  $TS'$  and  $R'$  respectively by corresponding runs. By *succ<sub>buff</sub>* for states and regions,  $\forall p, q, m : \beta(p, q, m) = \beta_R(p, q, m)$ .

The condition for disallowing action  $q?p(m)$  in equivalent (and hence abstractly represented states) are identical:  $\beta(p, q, m) \leq 0$ . Therefore, whenever  $q?p(m)$  is enabled in a reachable state  $s$ ,  $q?p(m)$  is enabled in the corresponding region  $s_R$  and the resulting states and regions correspond to each other.

The  $B$ -boundedness carries over from  $TS(\mathcal{A})$  to  $R(\mathcal{A})$ . Therefore there exist only finitely many different vectors  $\mathbf{w}$  of discrete states, finitely many different timed regions  $\alpha$  and finitely many different buffer state functions  $\beta$  that are reachable. Therefore the set of reachable regions of  $R(\mathcal{A})$  is finite. □

Similar to timed automata, we have shown in the proof that the region automaton of  $\mathcal{A}$  accepts the same untimed language as its underlying transition system. Please observe that the  $B$ -boundedness serves to ensure the finiteness of the region automaton, and we get the following corollary.

**Corollary 8:** *Let  $\mathcal{A}$  be  $B$ -bounded MTA. The untimed language  $L$  that is accepted by  $\mathcal{A}$  is regular.* \*

In a next step, we are going to introduce reference clocks and time stamps to region automata. Since reference clocks are never reset, but their (unbounded) values matter for the definition of a timed run, we are about to introduce a possibly new source of infinity into the region automaton.

As the purpose of region semantics is to reduce the infinite state space of the underlying transition system of  $\mathcal{A}$  equivalently (in terms of reachability) to a transition system of finite size, time stamps have to be reformulated in terms of regions as well. In the semantics we have defined so far, time stamps originate from reference clocks. Therefore, although reference clocks are clock variables, their introduction into region automata requires some attention.

**Introducing Reference Clocks** Considering a timed automaton  $A + g$  with reference clock  $g$ , we have to observe that  $A + g$  imposes no maximal constant  $k_g$  against which  $g$  is compared. Furthermore, the values of  $g$  do matter for the purpose of time-stamping. This implies that the classical definition of region equivalence  $\sim_r$  cannot yield finitely many different clock regions only (see Sect. 2.7. We will address the implication of  $\sim_r$  having an infinite index later.

First we need to be able to talk about constraints of single clock variables in a region. By the definition of region equivalence, the clock valuations of a region agree on constraints  $x = k$ ,  $k < x < k + 1$ , or  $x > k_x$ , for every  $x \in C$ . It will become convenient to talk about these in the shape of intervals in  $\mathbb{R}_{\geq 0}$ , i.e.  $x \in [k; k]$ ,  $x \in (k; k + 1)$ , and  $x \in (k_x; \infty)$ , respectively.

Let  $D \subseteq C$  be a subset of clock variables. For a set  $\Sigma$  of clock valuations, we define  $\Pi_D(\Sigma) =_{def} \{\sigma|_D \mid \sigma \in \Sigma\}$ . Remembering that a clock region  $\alpha \in R(C)$  is a special set of clock valuations,  $\Pi_D(\alpha)$  is the restriction to clocks from  $D$ , and  $\Pi_D(\alpha)$  is a clock region over  $D$ . For  $x \in C$ , we define  $\alpha(x) =_{def} \Pi_{\{x\}}(\alpha)$ . For any  $\alpha \in R(C)$  and any  $x \in C$  either there exists a  $k \in \mathbb{N}$  such that for all  $\sigma \in \alpha$  either  $\sigma(x) \in [k; k]$  or  $\sigma(x) \in (k; k + 1)$  holds, or  $\sigma(x) > k_x$  for all  $\sigma \in \alpha$ . Hence we may write  $\alpha(x) = [k; k]$  or  $\alpha(x) = (k; k + 1)$  or  $\alpha(x) = (k_x; \infty)$  accordingly. Note that the converse is not true in general as there are several regions  $\alpha, \alpha' \in R(C)$  satisfying  $\alpha(x) = \alpha'(x)$  for all  $x \in C$ , but that disagree on difference constraints (e.g.  $x - y < k$ ) of different clocks.

We are now able to talk about the “value” of a clock variable in a clock region and hence about the “value” of a reference clock. We can therefore move on to time stamps where we will use the term *region stamp* when speaking of time stamps in the region semantics.

Just like  $\sigma(g)$ , for some reference clock  $g$ , served as the time stamp of an action in the time-stamped clock semantics, we use  $\alpha(g)$  to region-stamp actions in the region semantics. The value of  $g_p$  is not bounded by  $\mathcal{A}$ . Therefore, the set of region stamps is the set of intervals  $RegionStamps =_{def} \{[k; k], (k; k + 1) \mid k \in \mathbb{N}\}$ . If  $g$  is a reference clock in a timed automaton  $A$ , then in any region  $(w, \alpha) \in RS(A)$ ,  $\alpha(g) \in RegionStamps$ .

By the linear order over  $\mathbb{R}$ , we can compare two region stamps  $\tau, \tau' \in RegionStamps$  as intervals:  $\tau < \tau'$  iff  $\forall t \in \tau, t' \in \tau' : t < t'$ . For example:  $[k; k] < (k; k + 1)$ . In the following, we will compare region stamps and values of reference clocks in a timed region.

Let  $\mathcal{A}^+$  be a  $MTA^+$  with reference clocks  $\mathbf{g} = (g_1, \dots, g_P)$  of the respective components. We region-stamp messages with values from  $RegionStamps$ . Thus the *region-*

*stamped buffer state function* in the region automaton of  $\mathcal{A}^+$  is a mapping from  $\mathcal{P} \times \mathcal{P} \times M$  to  $\mathcal{M}(\text{RegionStamps})$ .

Let  $\mathbf{w} \in W_1 \times \dots \times W_P$ ,  $\alpha \in R(C^+)$  and  $\beta$  a region-stamped buffer state function. Then  $(\mathbf{w}, \alpha, \beta)$  is a *stamped region* of  $\mathcal{A}^+$ , the set of all stamped regions being  $RS^{\mathbf{g}}(\mathcal{A}^+)$ . The *initial* stamped region  $s^0$  is straight forward.

Again, we define the operator  $\text{succ}_{\text{buff}}$  to obtain the buffer state that results from firing action  $a$  at region stamp  $\tau \in \text{RegionStamps}$  on a given buffer state  $\beta$ :  $\text{succ}_{\text{buff}}(a, \tau, \beta) = \beta'$  where

$$\beta'(p, q, m) =_{\text{def}} \begin{cases} \beta(p, q, m) + \tau, & \text{if } a = p!q(m) \in \Gamma_p^M \\ \beta(p, q, m) - \tau', & \text{if } a = q?p(m) \in \Gamma_q^M, \\ & \tau' \in \beta(p, q, m), \tau' \leq \tau \\ \beta(p, q, m), & \text{otherwise} \end{cases}$$

$\text{succ}_{\text{buff}}$  is well-defined because all region stamps in  $\text{RegionStamps}$  are comparable by  $<$  and  $=$ .

**Definition 14 (Region-Stamped Semantics of MTA<sup>+</sup>).** Let  $\mathcal{A}^+$  be a MTA<sup>+</sup> with reference clocks  $\mathbf{g} = (g_1, \dots, g_P)$ . Let  $s = ((w_1, \dots, w_P), \alpha, \beta) \in RS^{\mathbf{g}}(\mathcal{A}^+)$  be a stamped region.

1. The stamped region  $s' = ((w'_1, \dots, w'_P), \alpha', \beta') \in RS(\mathcal{A}^+)$  is reachable from  $s$  by a *discrete step*  $s \xrightarrow{a} s'$  iff
  - a)  $a \in \Gamma_p$ ,
  - b)  $w_p \xrightarrow{\varphi, a, R} w'_p$ ,  $\alpha \models \varphi$ ,  $\alpha' = \alpha[R \mapsto 0]$ ,  $\alpha' \models \text{inv}(w')$   
(this implies the discrete step  $(w_p, \Pi_{C_p^+}(\alpha)) \xrightarrow{a} (w'_p, \Pi_{C_p^+}(\alpha'))$  in  $R(A_p^+)$ ),
  - c) if  $a = p?q(m) \in \Gamma_p^?$  then  $\exists \tau' \in \beta(q, p, m) : \tau' \leq \alpha(g_p)$  holds,
  - d)  $\beta' = \text{succ}_{\text{ch}}(a, \alpha(g_p), \beta)$ , and
  - e) for all  $q \in \mathcal{P}$ ,  $p \neq q$  implies  $w'_q = w_q$
2. The stamped region  $s' = ((w'_1, \dots, w'_P), \alpha', \beta') \in RS(\mathcal{A}^+)$  is reachable from  $s$  by a *time step*  $s \xrightarrow{\delta} s'$  iff  $\alpha' \in \text{succ}_r^*(\alpha)$ ,  $w'_p = w_p$  and for all  $\alpha'' \in \text{succ}_r^*(\alpha)$  with  $\alpha' \in \text{succ}_r^*(\alpha'')$ ,  $\alpha' \models \text{inv}_p(w_p)$  for all  $p \in \mathcal{P}$ , and  $\beta' = \beta$   
(which implies the time-steps  $(w_p, \Pi_{C_p^+}(\alpha)) \xrightarrow{\delta} (w'_p, \Pi_{C_p^+}(\alpha'))$  or  $(w_p, \Pi_{C_p^+}(\alpha)) = (w'_p, \Pi_{C_p^+}(\alpha'))$  in  $R(A_p^+)$  for all  $p \in \mathcal{P}$ )
3. The stamped region  $s'$  is reachable from  $s$  by a *combined step*  $s \xrightarrow{\delta, a} s'$  iff there ex. a stamped region  $s''$  such that  $s \xrightarrow{\delta} s''$  and  $s'' \xrightarrow{a} s'$ . \*

The region automaton  $R^{\mathbf{g}}(\mathcal{A}^+)$  ( $R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  for combined steps) with regions  $RS^{\mathbf{g}}(\mathcal{A}^+)$  follows directly from Definition 14. Since  $R^{\mathbf{g}}(\mathcal{A}^+)$  contains reference clocks being the only source of global time information, the definition of a timed run of  $R^{\mathbf{g}}(\mathcal{A}^+)$  follows directly from Definition 11 of a timed run of  $TS^{\mathbf{g}}(\mathcal{A}^+)$  in Sect. 2.5. We region-stamp an

action with the value of the corresponding reference clock in the region that is reached by firing this action. In  $\rho = s^0(a_1, \tau_1)s^1(a_2, \tau_2)s^2 \dots$  with  $s^i = (\mathbf{w}^i, \alpha^i, \beta^i)$  for all  $i \geq 0$  holds:  $s_{i-1} \xrightarrow{\delta, a_i} s_i$  is a step in  $R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  and  $\tau_i = \alpha^i(g_p)$  where  $p = \text{loc}(a_i)$ . Observe that like in the time-stamped transition system  $TS^{\mathbf{g}}(\mathcal{A}^+)$  in all reachable states, the reference clocks in all components have the same value,  $\forall i \geq 0 \forall p, q \in \mathcal{P} : \alpha^i(g_p) = \alpha^i(g_q)$ .

Let  $Run_R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  denote the set of all region-stamped runs of  $\mathcal{A}^+$ . Similarly the set of *region-stamped ta-sequences*  $TaSeq_R^{\circ, \mathbf{g}}(\mathcal{A}^+)$ , the reached state ( $state_R^{\circ, \mathbf{g}}(\rho)$ ), and enabled actions ( $enabled_R^{\circ, \mathbf{g}}$ ) carry over from the concrete time-stamped semantics as given in Definition 11.

Observe that in analogy to Proposition 4:

**Proposition 9:** *For a given network of message passing timed automata  $\mathcal{A}$  with corresponding  $MTA^+ \mathcal{A}^+$  with reference clocks  $\mathbf{g}$ , the region automata  $R(\mathcal{A})$  and  $R^{\mathbf{g}}(\mathcal{A}^+)$  are bisimilar.* \*

*Proof.* We construct the bisimulation relation.

Consider  $t = (\mathbf{w}, \alpha, \beta) \in RS(\mathcal{A})$  and  $s = (\mathbf{w}^+, \alpha^+, \beta^+) \in RS^{\mathbf{g}}(\mathcal{A}^+)$ . We define the following predicate *sim* which holds if  $s$  and  $t$  coincide on the discrete states, if in  $t$  all reference clocks are equal and  $s$  and  $t$  coincide on the clock region, and if the message buffers in  $s$  and  $t$  correspond to each other.

$$\begin{aligned} sim(t, s) \quad =_{def} \quad & t = (\mathbf{w}, \alpha, \beta) \in RS(\mathcal{A}) \\ & \wedge s = (\mathbf{w}^+, \alpha^+, \beta^+) \in RS^{\mathbf{g}}(\mathcal{A}^+) \wedge \alpha = \Pi_C(\alpha^+) \\ & \wedge \forall p, q \in \mathcal{P} : \alpha^+(g_p) = \alpha^+(g_q) \\ & \wedge \forall m \in M(p, q) : |\beta^+(p, q, m)| = \beta(p, q, m) \\ & \wedge \forall \tau \in \beta^+(p, q, m) : \tau \leq \alpha^+(g_p) \end{aligned} \quad (1)$$

Assume that for the chosen  $t$  and  $s$ ,  $sim(t, s)$  holds.

First, we show that  $\mathcal{A}^+$  simulates  $\mathcal{A}$ . Let  $s'$  be a state reached from  $s$  by a discrete step in  $R^{\mathbf{g}}(\mathcal{A}^+)$ :  $s \xrightarrow{a} s_1$ .

Reference clock do not occur in guards or invariants and hence cannot enable or disable actions. The actions, guards and invariants of  $\mathcal{A}$  and  $\mathcal{A}^+$  are identical. Additionally, in  $R^{\mathbf{g}}(\mathcal{A}^+)$  messages are region-stamped and may be received in  $s$  iff the message's region-stamp is less or equal the receivers reference clock. Since  $sim(t, s)$  holds, all time-stamps are less or equal than the current reference clock and hence the condition for an action  $q?p(m)$  to receive a message reduces to  $|\beta^+(p, q, m)| > 0$ . This condition is satisfied in  $s$  iff the condition  $\beta(p, q, m) > 0$  is satisfied in  $t$ .

Hence action  $a$  is enabled in  $t$  and  $t \xrightarrow{a} t_1$  is a discrete step in  $R(\mathcal{A})$ . By the definition of steps and the definition of the *succ<sub>buff</sub>* operator for region-stamped buffer state functions and unstamped ones,  $sim(t_1, s_2)$  holds.

Now let  $s_2$  be a state reached from  $s$  by a time step in  $R^{\mathbf{g}}(\mathcal{A}^+)$ :  $s \xrightarrow{\delta} s_2$ , i.e.  $\alpha_2^+ \in succ_r^*(\alpha^+)$ . Again, by not appearing in invariants, the presence of reference clocks does not influence the progress of time in a state. Furthermore observe that the region-successor operation is defined such that  $succ_r^*(\alpha) = \{\Pi_C(\alpha^*) \mid \alpha^* \in succ_r^*(\alpha^+)\}$

holds. Therefore a time step  $t \xrightarrow{\delta} t_2$  is enabled such that  $\alpha_2 = \Pi_C(\alpha_2^+)$ . Consequently,  $\text{sim}(t_2, s_2)$  holds.

To show the opposite, that  $\mathcal{A}$  simulates  $\mathcal{A}^+$ , we remark that for discrete steps  $t \xrightarrow{a} t_1$  in  $R(\mathcal{A})$  by the same arguments as given above, a step  $s \xrightarrow{a} s_1$  in  $R^{\mathbf{g}}(\mathcal{A})$  exists such that  $\text{sim}(t_1, s_1)$  holds. Similarly, for time steps  $t \xrightarrow{\delta} t_2$  in  $R(\mathcal{A})$  exists a clock region  $\alpha_2^+ \in \text{succ}_r^*(\alpha)$  with  $\alpha_2 = \Pi_C(\alpha_2^+)$ ; we get a step  $s \xrightarrow{\delta} s_2$  in  $R^{\mathbf{g}}(\mathcal{A})$  with  $\text{sim}(t_2, s_2)$ .

The arguments above have shown that the predicate  $\text{sim}$  characterizes a bisimulation relation between  $R(\mathcal{A})$  and  $R^{\mathbf{g}}(\mathcal{A}^+)$ . Since  $\text{sim}$  holds on the respective initial regions, both region automata are bisimilar.  $\square$

It is worth noting that we defined  $\text{sim}$  in (1) in such a way that *every* region in  $s \in RS^{\mathbf{g}}(\mathcal{A}^+)$  has a corresponding region  $t \in RS(\mathcal{A})$ , and vice versa. Because the initial regions are related by  $\text{sim}$ , we can conclude the following nice result:

**Corollary 10:** *Let  $s \in RS^{\mathbf{g}}(\mathcal{A}^+)$ . Then there exists  $t \in RS(\mathcal{A})$  with  $\text{sim}(t, s)$ . Furthermore,  $t$  is reachable in  $R(\mathcal{A})$  iff  $s$  is reachable in  $R^{\mathbf{g}}(\mathcal{A}^+)$ .*  $\star$

The proof of bisimilarity already suggests our next step, to show that the region-stamped automaton and time-stamped transition system of the same  $\text{MTA}^+$  are related to each other by time-abstract bisimulation (just as are the region automaton and the normal transition system). The critical step in the proof is to show that a region-stamped message can be received iff a time-stamped can be received (in bisimilarly related states).

**Relating Region-Stamped and Time-Stamped Messages.** To relate time-stamped semantics and region-stamped semantics, we need to related time stamps and region stamps first. A time stamp  $t \in \mathbb{R}_0^+$  corresponds to a region stamp  $\tau \in \text{RegionStamps}$  iff  $t \in \tau$ .

$$\text{abstr}(t) =_{\text{def}} \begin{cases} [t; t], & t \in \mathbb{N} \\ ([t]; [t] + 1), & \text{otherwise} \end{cases}$$

Let  $\text{abstr}$  extend canonically to multisets of time stamps ( $\mathbb{R}_0^+$ ) and  $\text{RegionStamps}$  and from thereon to buffer-state functions.

Let  $\text{abstr} : S(C) \rightarrow R(C)$  be the natural abstraction function from clock valuations to clock regions that maps each clock valuation  $\sigma$  to the unique region  $\alpha$  containing  $\sigma$ :  $\text{abstr}(\sigma) = \alpha$  iff  $\sigma \in \alpha$ . We extend  $\text{abstr}$  to states by  $\text{abstr}(\mathbf{w}, \sigma, \beta) =_{\text{def}} (\mathbf{w}, \text{abstr}(\sigma), \text{abstr}(\beta))$ , to time-stamped actions  $\text{abstr}(a, t) =_{\text{def}} (a, \text{abstr}(t))$ , and canonically to runs and ta-sequences. Observe that  $\text{abstr}$  defines a homomorphism from time-stamped transition systems to stamped region automata of  $\text{MTA}^+$ .

**Lemma 11:** *Let  $\mathcal{A}^+$  be a  $\text{MTA}^+$  with reference clocks  $\mathbf{g} = (g_1, \dots, g_P)$ . For every concrete run  $\rho \in \text{Run}^{\circ, \mathbf{g}}(\mathcal{A}^+)$ :  $\text{abstr}(\rho) \in \text{Run}_R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  and for every abstract run  $\bar{\rho} \in \text{Run}_R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  exists a concrete run  $\rho \in \text{Run}^{\circ, \mathbf{g}}(\mathcal{A}^+)$  s.t.  $\bar{\rho} = \text{abstr}(\rho)$ .*  $\star$

*Proof.* For non-communicating actions  $s \xrightarrow{\delta, a} s'$ ,  $a \in \Gamma_p^I$ , the arguments of the proof of Proposition 7 hold here as well. Hence for the proof, it is sufficient to show that a region-stamped message can be received iff a time-stamped message can be received.

For concrete runs  $\rho \in \text{Run}^{\circ, \mathbf{g}}(\mathcal{A}^+)$  the proposition trivially holds by Prop. 7 and the fact that for all time steps  $t_i, t_j \in \mathbb{R}_0^+$  with  $t_i \leq t_j$  the corresponding region-stamps satisfy  $\text{abstr}(t_i) \leq \text{abstr}(t_j)$ . Thus any message that is receivable in concrete semantics is also receivable in abstract semantics.

Conversely, by abstracting time-stamps of sender  $t_i$  and receiver  $t_j$  to region stamps  $\tau_i$  and  $\tau_j$  in a run, the following cases can occur:

1.  $\tau_j < \tau_i$  and the message can neither be received in concrete, nor in abstract semantics.
2.  $\tau_i < \tau_j$  and the message can be received in abstract and concrete semantics.
3.  $\tau_i = [k; k] = \tau_j$  and the message can be received in abstract and concrete
4.  $\tau_i = (k; k + 1) = \tau_j$ . Then for every abstract run

$$\bar{\rho} = \bar{s}^0(a_1, \tau_1) \bar{s}^1(a_2, \tau_2) \dots$$

by Proposition 7, a send action  $a_i = p!q(m)$  is fireable in  $\bar{s}^i$  at  $\tau_i$  iff  $p!q(m)$  is fireable in  $s^i$  (with  $\text{abstr}(s^i) = \bar{s}^i$ ) at  $t_i \in \tau_i$ , by the definition of  $\text{abstr}$ .

The subsequent receive action  $a_j = q?p(m)$  with  $j > i$  is fireable in  $\bar{s}^j$  at  $\tau_j = \tau_i$  iff  $q?p(m)$  is fireable in  $s^j$  (with  $\text{abstr}(s_j) = \bar{s}_j$ ) at  $t_j \in \tau_j$ . If  $q?p(m)$  wasn't fireable then either because the action's guard does not hold which contradicts  $\text{abstr}(s_j) = \bar{s}_j$  or because the  $t_i > t_j$  which contradicts  $i < j$ .  $\square$

**Theorem 12.** *Let  $\mathcal{A}^+$  be a  $\text{MTA}^+$  with reference clocks  $\mathbf{g} = (g_1, \dots, g_P)$ . A region  $(\mathbf{w}, \alpha, \beta) \in \text{RS}(\mathcal{A}^+)$  is reachable in  $R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  iff there exists  $\sigma \in \alpha$  and a time-stamped buffer state function  $\tilde{\beta}$  with  $\beta = \text{abstr}(\tilde{\beta})$  such that  $(\mathbf{w}, \sigma, \tilde{\beta})$  is reachable in  $\text{TS}^{\circ, \mathbf{g}}(\mathcal{A}^+)_*$*

*Proof.* The proposition follows from the definition of reachable state and reachable region (see Def. 6) and the homomorphism  $\text{abstr}$  from runs of  $\text{TS}^{\circ, \mathbf{g}}(\mathcal{A}^+)$  to runs of  $R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  shown in Lemma 11.  $\square$

### 2.8.1 Time Stamping Messages with Zones

We can now give a reason for choosing regions instead of zones to symbolically represent infinite sets of clock valuations. Some knowledge about clock zones and the zone automaton [AD94], will facilitate to understand the explanation, but an intuitive understanding should be sufficient to get the main argument.

We just introduced region-stamped semantics, where a message is stamped with the value of the reference clock of the sending component in the moment of sending. This information is necessary in the local semantics to ensure that a message is received only

if the receiving action  $q?p(m)$  also occurs timely after the sending action  $p!q(m)$ . We showed in Lemma 11 that stamping messages with the region constraint  $\alpha(g_p)$  of the sender's reference clock is sufficient to guarantee this property.

If we applied the zone abstraction, it was natural to do something similar and stamp each message with the zone's clock constraint for the reference clock. Unfortunately, this clock constraint can be too large: It may happen that the guards of sending and receiving actions are such that the receiving action gets disabled before the sending action. See Fig. 11 for a minimal example.

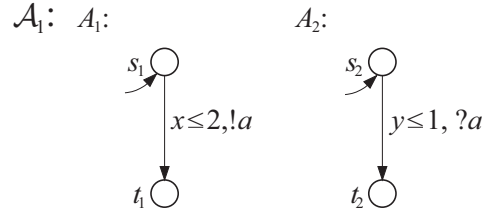


Figure 11: A MTA where the receiving action  $?a$  gets disabled before the sending action  $!a$ .

Assume some local semantics using zones, where the clock valuations are abstracted componentwise: each abstract state has one clock zone per component. The zone abstraction for  $A_1^+$  in the MTA<sup>+</sup>  $\mathcal{A}_1^+$  generates the information that  $!a$  may send<sup>8</sup> a message any time between in  $0 \leq x = g_1 \leq 2$ , where  $g_1$  is the reference clock of  $A_1^+$ . The available information in  $A_1$  allows us to stamp the message with  $[0; 2]$ . The zone abstraction for  $A_2$  generates the information that  $?a$  may receive messages any time between  $0 \leq y = g_2 \leq 1$ . Of course, the intervals of sender and receiver are overlapping and hence there is a time where the message can be received. But by a “zone-stamp”  $[0; 2]$  we cannot distinguish the moments of time where in concrete semantics the message can be received from the moments of time where they cannot be received. This violates the idea of the zone abstraction where a zone represents sets of clock valuations that yield equivalent behaviour.

The region abstraction provides five different region stamps ( $[0; 0]$ ,  $(0; 1)$ ,  $[1; 1]$ ,  $(1; 2)$ , and  $[2; 2]$ ) for the messages in the system in Fig. 11 and hence distinguishes these moments in time. To apply zone abstraction, it would be necessary to split the zone of the sender at  $x = g_1 = 1$ . We consider this as future work and confine ourselves for the moment to the directly applicable abstraction using clock regions.

## 2.9 Local-Region Semantics

Up to now, we have done a lot of preparatory work to facilitate the upcoming proofs. Currently, we have defined a local-clock semantics of MTA in the spirit of Bengtsson et al. [BJLY98]. We will now do the first new step towards our desired result of a finite

<sup>8</sup> $!a$  is used as a short-cut notation for  $1!2(a)$ .

representation of the state space of a timed automaton (through regions) *and* to have concurrently enabled actions to be independent; we will define *local-region semantics* of MTA.

As we have already seen in the introduction in Sect. 1.3, the independence result in a finite representation ought to be achieved much easier if the abstract state of the system can be distributed, just as we did in the local clock semantics in Sect. 2.6.

As in local clock semantics, we take a straight approach and decompose a region of the system into parts that correspond directly to each of the system's components.

Let  $s = ((w_1, \dots, w_P), \alpha, \beta)$  be a stamped region of  $\mathcal{A}^+$ . By

$$\pi_{\mathcal{P}}(s) =_{def} ((w_1, \Pi_{C_1^+}(\alpha)), \dots, (w_P, \Pi_{C_P^+}(\alpha)), \beta)$$

we denote the *decomposed region* of  $s$ . Clearly,  $\pi_{\mathcal{P}}(s)$  contains less information than  $s$  as the relations of clocks in different components are lost. By  $RS_t^{\mathbf{g}}(\mathcal{A}^+) =_{def} \pi_{\mathcal{P}}(RS(\mathcal{A}^+))$  denote the set of all decomposed regions of  $\mathcal{A}^+$ .

The notion of a step will follow the idea for local-clock semantics: advance time locally and perform actions locally, which is possible on the decomposed regions. However, we need to pay attention on the local steps.

In Corollary 8 we postulated that the untimed language that is accepted by a  $B$ -bounded MTA is regular. Lemma 1 undermines that we need to be careful in the choice of the enabledness of local actions not to lose this property; an ill-chosen enabledness could yield an independence relation by which the trace-closure of the accepted language is not regular. To be on the safe side, we want to ensure a finite representability of the state space, even in local-region semantics.

Please consider the system  $\mathcal{A}_2$  in Fig. 12, where the system accepts the untimed language  $L = \{(p!q(m) q?p(m))^n \mid n \in \mathbb{N}\}$ .  $\mathcal{A}_2$  is 1-bounded by its guards and invariants.

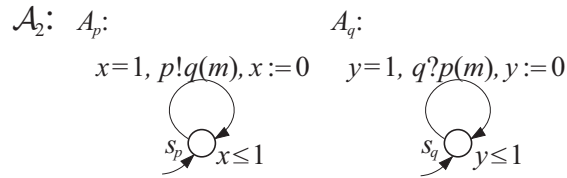


Figure 12: A  $B$ -bounded MTA where arbitrarily local actions create infinitely many different buffer states.

If we allowed  $p!q(m)$  to occur based on the enabledness of its guard only, we would generate runs where an arbitrary number of messages  $m$  is produced before a message is consumed. Allowing such runs would allow infinitely many different buffer states. We could not ensure a finite representability of the state space in local-region semantics. To maintain this property, we have to restrict the enabledness of send-actions.

Considering  $B$ -bounded systems, we cannot let a send action occur in local semantics if its buffer is already filled with  $B$  messages. This will not reduce the set of reachable states in comparison with the original semantics, in which there is no send action adding a  $B + 1$ st message to the buffer.

**Definition 15 (Local Region Semantics of  $\text{MTA}^+$ ).** Let  $\mathcal{A}^+$  be a  $B$ -bounded  $\text{MTA}^+$  with reference clocks  $(g_1, \dots, g_P)$  and let  $s = ((w_1, \alpha_1), \dots, (w_P, \alpha_P)), \beta \in RS_l^{\mathbf{g}}(\mathcal{A}^+)$  be a decomposed region.

1. The decomposed region  $s' = ((w'_1, \alpha'_1), \dots, (w'_P, \alpha'_P)), \beta' \in RS_l^{\mathbf{g}}(\mathcal{A}^+)$  is reachable from  $s$  by a *discrete step* of  $p \in \mathcal{P}$ ,  $s \xrightarrow{a}_p s'$  iff
  - a)  $a \in \Gamma_p$ ,
  - b)  $(w_p, \alpha_p) \xrightarrow{a}_p (w'_p, \alpha'_p)$ ,
  - c) if  $a = p?q(m) \in \Gamma_p^?$  then  $\exists \tau' \in \beta(q, p, m) : \tau' \leq \alpha(g_p)$  holds,
  - d) if  $a = p!q(m) \in \Gamma_p^!$  then  $|\beta(p, q, m)| < B$  holds,
  - e)  $\beta' = \text{succ}_{ch}(a, \alpha(g_p), \beta)$ ,
  - f) for all  $q \in \mathcal{P}$ ,  $p \neq q$  implies  $w'_q = w_q$  and  $\alpha'_q = \alpha_q$ .
2. The decomposed region  $s' = ((w'_1, \alpha'_1), \dots, (w'_P, \alpha'_P)), \beta' \in RS_l^{\mathbf{g}}(\mathcal{A}^+)$  is reachable from  $s$  by a *time step*  $s \xrightarrow{\delta}_p s'$  iff  $(w_p, \alpha_p) \xrightarrow{\delta}_p (w'_p, \alpha'_p)$ , and for all  $q \in \mathcal{P}$  holds  $w'_q = w_q$  and  $q \neq p$  implies  $\alpha'_q = \alpha_q$ ,  $\beta' = \beta$ .
3. The stamped region  $s'$  is reachable from  $s$  by a *combined step*  $s \xrightarrow{a} s'$  iff there ex.  $p \in \mathcal{P}$  and a stamped region  $s''$  such that  $s \xrightarrow{\delta}_p s''$  and  $s'' \xrightarrow{a}_p s'$ . \*

Observe that local region semantics, as we have given it here, is defined for  $B$ -bounded  $\text{MTA}^+$  only. The corresponding region automaton – we call it *local region automaton* –  $R_l^{\mathbf{g}}(\mathcal{A}^+)$  ( $R_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$  for combined steps) as well as *local-region runs* ( $\text{Run}_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+)$ ), *local-region ta-sequences* ( $\text{TaSeq}_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+)$ ), and the thereby reached state ( $\text{state}_{R,l}^{\circ, \mathbf{g}}(\rho)$ ) as well as enabledness of actions ( $\text{enabled}_{R,l}^{\circ, \mathbf{g}}$ ) is straight forward (see Definition 11).

### 2.9.1 Independence in the Local Region Graph

The local region automaton describes the system's behaviour in terms of local actions that sequentially modify different parts of the state. Additionally, in the local region semantics the result of firing an action depends only on the parts of the state it may modify in turn. Thus this kind of locality is sufficient for a notion of independence which we will examine now.

Some preparatory work needs to be done first. Up to now we defined runs through combined steps  $s \xrightarrow{a} s'$  but the proofs of independence of actions and subsequent proofs of correctness suggest a notion of run defined through local discrete steps and local time steps. Yet, the definition of such a labelled run is straight forward as a way in  $R_l^{\mathbf{g}}(\mathcal{A}^+)$  (having *no* composed steps) while we require that a time step now must let time pass (i.e.  $s \xrightarrow{\delta} s$  is not allowed).

**Definition 16 (Non-Composed Local Region Run of  $\text{MTA}^+$ ).** A *non-composed local region run* of an  $\text{MTA}^+$   $\mathcal{A}^+$  with reference clocks  $\mathbf{g} = (g_1, \dots, g_P)$  is an alternating

sequence of decomposed regions and region-stamped actions  $(a, \tau)$  or region-stamped time-passes  $(p, \tau)$  of components  $p \in \mathcal{P}$ :  $\rho = s^0(\omega_1, \tau_1)s^1(\omega_2, \tau_2)\dots$ , where for all  $i > 0$  with  $s^i = ((w_1^i, \alpha_1^i), \dots, (w_p^i, \alpha_p^i), \beta^i) \in RS_l^{\mathbf{g}}(\mathcal{A}^+)$  and  $(\omega_i, \tau_i) \in (\Gamma \cup \mathcal{P}) \times \text{RegionStamps}$ :

- if  $\omega_i \in \Gamma$  (an action) then  $\tau_i = \alpha_p^i(g_p)$  with  $p = \text{loc}(\omega_i)$ , and  $s^{i-1} \xrightarrow{\omega_i} s^i$  is a local discrete step of  $p$  in  $R_l^{\mathbf{g}}(\mathcal{A}^+)$ ,
- if  $\omega_i \in \mathcal{P}$  (a time-pass) then  $\tau_i = \alpha_p^i(g_p)$  with  $p = \omega_i$ , and  $s^{i-1} \xrightarrow{\delta} s^i$  is a local time step of  $p$  with  $s^{i-1} \neq s^i$  in  $R_l^{\mathbf{g}}(\mathcal{A}^+)$ .

Let  $\text{Run}_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$  denote the set of all *non-composed local region runs* of  $\mathcal{A}^+$ , and correspondingly  $\text{TaSeq}_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$  the set of all *non-composed local region ta-sequences* of  $\mathcal{A}^+$  (see Definition 11). Similar to Definition 11,  $\text{state}_{R,l}^{\mathbf{g}}(\rho)$  denotes the state that is reached in  $R_l^{\mathbf{g}}(\mathcal{A}^+)$  through a given run or ta-sequence  $\rho$ . \*

Obviously, the sets  $\text{Run}_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$  and  $\text{Run}_{R,l}^{\circ,\mathbf{g}}(\mathcal{A}^+)$  relate to each other in the same way as local discrete steps and local time steps relate to local composed steps.

We will now give a syntactic characterization of independent actions in terms of a symmetric relation and then show that this characterization yields semantic independence (in local-region semantics): any two (syntactically) *independent actions* can neither disable nor enable each other and any order of execution yields the same successor state.

The independence relation is context-sensitive wrt a state  $s \in RS_l^{\mathbf{g}}(\mathcal{A}^+)$  and applicable to  $B$ -bounded MTA only: two actions are independent in  $s$  if they occur in different components and in case it is a matching pair of send- and receive actions,  $p!q(m)$  and  $q?p(m)$ , the buffer  $(p, q, m)$  is not empty and contains less than  $B$  messages. The latter criterion on communicating actions is necessary to ensure the enabledness of either action regardless of which action is occurring first.

For the following proofs it turns out to be more convenient if we consider a context-criterion based on ta-sequences. Every reachable state  $s$  in  $R_l^{\mathbf{g}}(\mathcal{A}^+)$  has a ta-sequence  $\hat{\rho} \in \text{TaSeq}_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$  leading to it ( $\text{state}_{R,l}^{\mathbf{g}}(\hat{\rho}) = s$ ). From the structure of  $\text{MTA}^+$  follows that the number of messages in the buffer  $(p, q, m)$  in  $s$  is the difference of numbers of send actions  $p!q(m)$  and of receive actions  $q?p(m)$  in  $\hat{\rho}$ .

**Definition 17 (Independent Actions).** Let  $\hat{\rho} \in \text{TaSeq}_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$  be a ta-sequence and let for  $p, q \in \mathcal{P}$ ,  $\omega \in \Gamma_p \cup \{p\}$ ,  $\psi \in \Gamma_q \cup \{q\}$  be two actions. The ta-sequence sensitive *dependency relation*  $D(\hat{\rho})$  is given by:  $(\omega, \psi) \in D(\hat{\rho})$  iff  $p = q$ , or  $\omega = p!q(m)$  and  $\psi = q?p(m)$  and  $0 < |\Pi_{\Gamma}(\hat{\rho})|_{\omega} - |\Pi_{\Gamma}(\hat{\rho})|_{\psi} < B$ . The ta-sequence sensitive *independence relation*  $I(\hat{\rho})$  is its complement:  $(\omega, \psi) \in I(\hat{\rho})$  iff  $(\omega, \psi) \notin D(\hat{\rho})$ . \*

This independence relation (without the upper bound  $B$  for the channel) was found before for netcharts in [MKT03]. Our extension also yields independent actions in the case of local-region semantics.

**Proposition 13:** Let  $\hat{\rho} \in \text{TaSeq}_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$  be a ta-sequence with  $s = \text{state}_{R,l}^{\mathbf{g}}(\hat{\rho})$  and for  $p, q \in \mathcal{P}$  let  $\omega \in \Gamma_p \cup \{p\}$ ,  $\psi \in \Gamma_q \cup \{q\}$  be two actions. Furthermore let  $\tau, v \in$

*RegionStamps* and  $(\omega, \tau), (\psi, v) \in \text{enabled}_{R,l}^{\mathbf{g}}(s)$  yielding states  $s_\omega = \text{state}_{R,l}^{\mathbf{g}}(\hat{\rho}(\omega, \tau))$  and  $s_\psi = \text{state}_{R,l}^{\mathbf{g}}(\hat{\rho}(\psi, v))$ .

If  $(\omega, \psi) \in I(\hat{\rho})$ , then  $(\psi, v)$  and  $(\omega, \tau)$  are independent:

1.  $(\omega, \tau) \in \text{enabled}_{R,l}^{\mathbf{g}}(s_\psi)$  and  $(\psi, v) \in \text{enabled}_{R,l}^{\mathbf{g}}(s_\omega)$ ,
2.  $\text{state}_{R,l}^{\mathbf{g}}(\hat{\rho}(\omega, \tau)(\psi, v)) = \text{state}_{R,l}^{\mathbf{g}}(\hat{\rho}(\psi, v)(\omega, \tau))$ . \*

*Proof.* The proof of the proposition is straight forward because enabledness of actions  $a$  of component  $p$  in a region depends on the discrete state of  $p$  and the clock region of  $p$  and the adjacent buffers only. Similarly, an action influences these parts of the state only. Our independence criterion guarantees that actions that are independent by  $I$  after executing some ta-sequence  $\hat{\rho}$  cannot influence each other and.  $\square$

We now need to move the independence result from ta-sequences of the region automaton of non-composed steps ( $TaSeq_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$ ) to ta-sequences of the region automaton of composed steps ( $TaSeq_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+)$ ). We do so by composing subsequent local time steps and local discrete steps into combined steps. Let  $\rho \in \text{Run}_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$ . W.l.o.g.  $\rho$  contains no two subsequent local time steps in the same component as they may be combined in a single larger time step. We may now introducing local time steps  $s \xrightarrow{\delta}_p s$  that let no time pass between any two subsequent local discrete steps of any component  $p$  and obtain a run where every local discrete step in any component is preceded by a local time step. Through repeatedly swapping a local time step of a component  $p$  with a discrete step of another component  $q$ , we obtain a run  $\rho^*$  where every local discrete step  $s'' \xrightarrow{a}_p s'$  is preceded directly by a local time step  $s \xrightarrow{\delta}_p s''$ , i.e. for every discrete step of any component  $p$  occurring in  $\rho^*$  exist a partition into subsequences  $\underline{\rho}^*, \overline{\rho}^*$  such that

$$\rho^* = \underline{\rho}^* s(p, \tau'') s''(a, \tau') s' \overline{\rho}^*, \quad (2)$$

where  $\text{loc}(a) = p$ . By the result of Proposition 13, the reached states are equal:  $\text{state}_{R,l}^{\mathbf{g}}(\rho) = \text{state}_{R,l}^{\mathbf{g}}(\rho^*)$ . Furthermore, in every discrete step  $s'' \xrightarrow{a}_p s'$ , the values of the reference clocks stay put ( $\forall q \in \mathcal{P} : \alpha_q''(g_q) = \alpha_q'(g_q)$ ), and local time steps in  $\rho$  and  $\rho^*$  are region-stamped with the reference clock of the reached region (see Definition 16), i.e.  $\tau'' = \tau'$  in every partition (2). Therefore each discrete step may be composed with its preceding time step and gets stamped with their common region stamp. We obtain the run  $\rho^\circ$  consisting of composed steps where the corresponding partition to (2) is

$$\rho^\circ = \underline{\rho}^\circ s(a, \tau') s' \overline{\rho}^\circ, \quad (3)$$

and  $\text{state}_{R,l}^{\circ, \mathbf{g}}(\rho^\circ) = \text{state}_{R,l}^{\mathbf{g}}(\rho^*)$  holds by the definition of composed steps. This transformation and the reachability result carries over to ta-sequences: for every ta-sequence  $\hat{\rho} \in TaSeq_{R,l}^{\mathbf{g}}(\mathcal{A}^+)$  with correspondingly transformed ta-sequence  $\hat{\rho}^\circ \in TaSeq_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+)$  holds  $\text{state}_{R,l}^{\circ, \mathbf{g}}(\hat{\rho}^\circ) = \text{state}_{R,l}^{\mathbf{g}}(\hat{\rho})$ , and vice versa.

The independence on actions extends to ta-sequences of composed steps in the sense that if two ta-sequences differ in independent actions only, their reached states are identical.

**Proposition 14:** Let  $\hat{\rho}_1, \hat{\rho}_2 \in TaSeq_{R,l}^{\circ,g}(\mathcal{A}^+)$  such that there exist a common prefix  $\underline{\rho}$  and a common suffix  $\bar{\rho}$  and actions  $a, b \in \Gamma$  with  $\hat{\rho}_1 = \underline{\rho}(a, \tau_a)(b, \tau_b)\bar{\rho}$ , and  $\hat{\rho}_2 = \underline{\rho}(b, \tau_b)(a, \tau_a)\bar{\rho}$ , and  $(a, b) \in I(\underline{\rho})$ . Then  $state_{R,l}^{\circ,g}(\hat{\rho}_1) = state_{R,l}^{\circ,g}(\hat{\rho}_2)$ .  $\star$

*Proof.* Follows from Proposition 13 and the transformation of runs given above.  $\square$

This result allows us to define an equivalence relation on ta-sequences where two ta-sequences are equivalent if they differ in independent actions only. The beautiful result that equivalent ta-sequences yield identical states then follows.

**Definition 18.** We define  $\sim_{tr} \subseteq (TaSeq_{R,l}^{\circ,g}(\mathcal{A}^+))^2$  to be the least equivalence relation such that for any ta-sequences  $\hat{\rho}_1, \hat{\rho}_2 \in TaSeq_{R,l}^{\circ,g}(\mathcal{A}^+)$  with prefix  $\underline{\rho}$  and suffix  $\bar{\rho}$  such that  $\hat{\rho}_1 = \underline{\rho}(a, \tau_a)(b, \tau_b)\bar{\rho}$ , and  $\hat{\rho}_2 = \underline{\rho}(b, \tau_b)(a, \tau_a)\bar{\rho}$ , the independence  $(a, b) \in I(\underline{\rho})$  implies  $\hat{\rho}_1 \sim_{tr} \hat{\rho}_2$ .

By  $[\hat{\rho}_1]_{tr}$  we denote the equivalence class of  $\hat{\rho}_1$  wrt  $\sim_{tr}$ .  $\star$

**Proposition 15:** Let  $\hat{\rho} \in TaSeq_{R,l}^{\circ,g}(\mathcal{A}^+)$  be a local-region ta-sequence. Then  $\forall \hat{\rho}_1, \hat{\rho}_2 \in [\hat{\rho}]_{tr} : state_{R,l}^{\circ,g}(\hat{\rho}_1) = state_{R,l}^{\circ,g}(\hat{\rho}_2)$ .  $\star$

*Proof.* Let  $\hat{\rho} \in TaSeq_{R,l}^{\circ,g}(\mathcal{A}^+)$  and  $\hat{\rho}_1, \hat{\rho}_2 \in [\hat{\rho}]_{tr}$ . By the definition of  $[\ ]_{tr}$  and  $\sim_{tr}$  exists  $n \in \mathbb{N}$  and ta-sequences  $\hat{\rho}_1, \dots, \hat{\rho}_n$  with  $\hat{\rho}_1 = \hat{\rho}_1$  and  $\hat{\rho}_n = \hat{\rho}_2$  and for all  $1 \leq i < n$  holds:  $\hat{\rho}_i$  and  $\hat{\rho}_{i+1}$  differ in the order of two subsequent independent actions only. Therefore for all  $1 \leq i < n$  by Prop. 14,  $state_{R,l}^{\circ,g}(\hat{\rho}_i) = state_{R,l}^{\circ,g}(\hat{\rho}_{i+1})$  and the proposition holds.  $\square$

By the result of Proposition 15, we may conceive an entire equivalence class  $[\hat{\rho}]_{tr}$  as a *trace* in local-region semantics since all member of the class yield the same state. We will speak of *trace equivalence* when talking about equivalent ta-sequences.

A *prime trace* is an equivalence class (of finite ta-sequences) where all ta-sequences end with the same region-stamped action  $(a, \tau_a)$ .

**Definition 19 (Prime Trace).** A trace  $[\hat{\rho}]_{tr}$ , with  $\hat{\rho} \in TaSeq_{R,l}^{\circ,g}(\mathcal{A}^+)$  is a *prime trace* if all ta-sequences  $\hat{\rho}_1, \hat{\rho}_2 \in [\hat{\rho}]_{tr}$  are such that  $\hat{\rho}_1 = \hat{\rho}_1(a, \tau_a)$  and  $\hat{\rho}_2 = \hat{\rho}_2(a, \tau_a)$ .<sup>9</sup> We write  $last([\hat{\rho}]_{tr}) =_{def} (a, \tau_a)$  for the last region-stamped action in  $[\hat{\rho}]_{tr}$  and  $pref([\hat{\rho}]_{tr}) =_{def} [\hat{\rho}_1]_{tr}$  for the equivalence class of all prefixes of  $[\hat{\rho}]_{tr}$ .  $\star$

## 2.9.2 Event Structure for Local-Region Semantics of $MTA^+$

From the notion of prime traces, we can define a labelled event structure of an  $MTA^+$  and obtain the nice result that the event structure expresses exactly the same behaviour as the local region automaton.

Prime traces will be the events in our event structure. Their causal relations follows from the prefix-relation on ta-sequences. To see that this prefix-relation can be used to order prime traces, we need an intermediary result.

---

<sup>9</sup>Observe that  $\hat{\rho}_1 \sim_{tr} \hat{\rho}_2$ .

**Lemma 16:** Let  $\hat{\rho}_1, \hat{\rho}_2 \in \text{TaSeq}_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+)$  be two ta-sequences of  $\mathcal{A}^+$  in local-region semantics. If  $\hat{\rho}_1 \sqsubseteq \hat{\rho}_2$  then for all  $\hat{\rho}'_1 \in [\hat{\rho}_1]_{tr}$  exists a  $\hat{\rho}'_2 \in [\hat{\rho}_2]_{tr}$  such that  $\hat{\rho}'_1 \sqsubseteq \hat{\rho}'_2$ .  $\star$

*Proof.* Let  $\hat{\rho}'_1 \in [\hat{\rho}_1]_{tr}$ . One can transform  $\hat{\rho}'_1$  into  $\hat{\rho}_1$  by swapping adjacent, independent actions. Because  $\hat{\rho}_1 \sqsubseteq \hat{\rho}_2$ , the reverse application of these swap operations yields a ta-sequence  $\hat{\rho}'_2$  with  $\hat{\rho}'_1 \sqsubseteq \hat{\rho}'_2$ . Because the swapped actions are independent (by the definition of  $I$ ),  $\hat{\rho}'_2 \sim_{tr} \hat{\rho}_2$  holds.  $\square$

For this reason, we may lift the prefix relation  $\sqsubseteq$  on sequences to equivalence classes: Given two traces  $[\hat{\rho}_1]_{tr}, [\hat{\rho}_2]_{tr}$ , we write  $[\hat{\rho}_1]_{tr} \sqsubseteq [\hat{\rho}_2]_{tr}$  if there exist  $\hat{\rho}'_1 \in [\hat{\rho}_1]_{tr}, \hat{\rho}'_2 \in [\hat{\rho}_2]_{tr}$  with  $\hat{\rho}'_1 \sqsubseteq \hat{\rho}'_2$ .

**Definition 20 (Event Structure of Local-Region Semantics of  $\text{MTA}^+$ ).** The labelled event structure  $ES^{\mathbf{g}}(\mathcal{A}^+) = (E^{\mathbf{g}}(\mathcal{A}^+), \leq, \#, \lambda)$  of an  $\text{MTA}^+$  with reference clocks  $\mathbf{g}$  is given by

- the set of *events* being the set of prime traces of  $\mathcal{A}^+$ :  $E^{\mathbf{g}}(\mathcal{A}^+) := \{[\hat{\rho}]_{tr} \mid \hat{\rho} \in \text{TaSeq}_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+), [\hat{\rho}]_{tr} \text{ is a prime trace}\}$ ,
- the *causality relation*  $\leq \subseteq (E^{\mathbf{g}}(\mathcal{A}^+))^2$  defined by:  $e \leq e'$  iff  $e \sqsubseteq e'$ ,
- the *minimal conflict relation*  $\# \subseteq (E^{\mathbf{g}}(\mathcal{A}^+))^2$  defined by:  
 $e \# e'$  iff  $\exists \hat{\rho} \in e, \hat{\rho}' \in e' : \hat{\rho} = \underline{\rho}(a, \tau_a)\bar{\rho} \wedge \hat{\rho}' = \underline{\rho}(b, \tau_b)\bar{\rho}' \wedge a \neq b \wedge (a, b) \notin I(\hat{\rho})$ , and
- the *labelling function*  $\lambda : E^{\mathbf{g}}(\mathcal{A}^+) \rightarrow \Gamma \times \text{RegionStamps}$  with  $\lambda(e) =_{def} \text{last}(e)$ .  $\star$

In case confusion with the  $\text{MTA}^+$  or the involved semantics can safely be excluded, we write  $ES$  and  $E$  instead of  $ES^{\mathbf{g}}(\mathcal{A}^+)$  and  $E^{\mathbf{g}}(\mathcal{A}^+)$ , respectively.

The *concurrency relation*  $\parallel$  follows from  $\leq$  and  $\#$  as usual:  $e \parallel e'$  iff neither  $e \leq e'$  nor  $e' \leq e$  nor  $e \# e'$ .

We show that in the event structure, events that are labelled with actions of the same component are either causally ordered, or conflicting. This proves that the event structure preserves the sequentiality of the components.

**Lemma 17:** Let  $e_a, e_b \in E$  with  $\lambda(e_a) = (a, \tau_a)$  and  $\lambda(e_b) = (b, \tau_b)$ . If  $\text{loc}(a) = \text{loc}(b)$  then  $e_a \leq e_b$  or  $e_b \leq e_a$  or  $e_a \# e_b$ .  $\star$

*Proof.* By definition of the dependency relation  $D$ ,  $(a, b) \in D(\hat{\rho})$  for all ta-sequences  $\hat{\rho}$ . Hence actions  $a$  and  $b$ , if occurring in the same trace are in fixed order in the ta-sequences's equivalence class, or are conflicting otherwise: for all  $\hat{\rho}_a \in e_a, \hat{\rho}_b \in e_b$  either  $\hat{\rho}_a \sqsubseteq \hat{\rho}_b$  or  $\hat{\rho}_b \sqsubseteq \hat{\rho}_a$  or there exists (possibly empty) subsequences  $\underline{\rho}, \bar{\rho}_a, \bar{\rho}_b$  such that  $\hat{\rho}_a = \underline{\rho}(a, \tau_a)\bar{\rho}_a, \hat{\rho}_b = \underline{\rho}(b, \tau_b)\bar{\rho}_b$ . Thus by definition of  $\leq$  and  $\#$  the proposition holds.  $\square$

A set  $K \subseteq E$  of events is a *configuration* if for all  $e \in K$  and all  $e' \in E$ ,  $e' \leq e$  implies  $e' \in K$  and no two events of  $K$  are in conflict. Each event  $e$  induces the unique *prime configuration*  $[e] =_{def} \{e' \in E \mid e' \leq e\}$ , its downward-closed subset wrt  $\leq$ . An event  $e$  is an *extension* of a configuration  $K$  if  $e \notin K$  and  $[e] \setminus \{e\} \subseteq K$ .

Given a configuration  $K$ , the set of its *linearizations*  $Lin(K)$  is the set of all causally ordered sequences of the elements of  $K$ :

$$Lin(K) =_{def} \{(e_1, \dots, e_n) \mid \{e_1, \dots, e_n\} = K, \forall 1 \leq i, j \leq n : e_i \leq e_j \Rightarrow i \leq j\}.$$

Observe that for every event  $e = [\hat{\rho}]_{tr}$ , every linearization  $\mathbf{e} = (e_1, \dots, e_n)$  of  $[e]$  represents a trace in  $e$ :  $\lambda(e_1) \dots \lambda(e_n) \in e$ . We will write  $\lambda(\mathbf{e})$  for  $\lambda(e_1) \dots \lambda(e_n)$ . The following result shows how traces and configurations are related to each other and that the event structure expresses *exactly* the same behaviour as the local region automaton.

**Proposition 18:** *Let  $\mathcal{A}^+$  be an  $MTA^+$  with reference clocks  $\mathbf{g}$ . Let  $K \subseteq E^{\mathbf{g}}(\mathcal{A}^+)$  be a configuration.*

1. For all  $\mathbf{e} \in Lin(K)$ ,  $\lambda(\mathbf{e}) \in TaSeq_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+)$ .
2.  $\{\lambda(\mathbf{e}) \mid \mathbf{e} \in Lin(K)\}$  is a trace equivalence class. \*

To prove the claims of Proposition 18, we show that concurrent events are labelled with independent actions. From thereon follows that the linearizations of a configuration yield ta-sequences. Because of the independency of concurrent events, the set of linearizations is complete wrt trace equivalence. The proof itself is rather technical.

*Proof.* induction on  $|K| = n$ . With  $n = 1$  the propositions are trivially satisfied. Let  $n > 1$  and  $(e_1, \dots, e_n) \in Lin(K)$ ,  $e_n \in K$  is a causally maximal event ( $\forall e' \in K : e_n \neq e' \Rightarrow \neg e_n \leq e'$ ) and  $K_n := K \setminus \{e_n\}$  the corresponding configuration with extension  $e_n$ . Let  $K' \subset K$  be any configuration. By inductive assumption,  $\{\lambda(\mathbf{e}') \mid \mathbf{e}' \in Lin(K')\} =: [\hat{\rho}(K')]_{tr}$  is a trace equivalence class with  $s(K') := state_{R,l}^{\circ, \mathbf{g}}(\hat{\rho}(K'))$  with  $\hat{\rho}(K') := \lambda(\mathbf{e}')$  for some  $\mathbf{e}' \in Lin(K')$ .

We first show that that all linearizations of  $K$  are ta-sequences. To achieve this we need to relate concurrent events to independent actions. We will examine two concurrent events first and extend our result later to sets of concurrent events by proving that concurrent events are labelled with independent actions. Through independence we can extend the ta-sequences of  $K_n$  to ta-sequences containing the actions of all events in  $K$ .

Consider two events  $f_0, f_1 \in K_n$  such that  $f_1$  is an extension of  $[f_0]$  and of  $L_0 := [f_0] \setminus \{f_0\}$ .

**Claim 18.1:**  $\lambda(f_0)$  and  $\lambda(f_1)$  are independent at  $\hat{\rho}(L_0)$ . \*

Obviously, neither  $f_0 \leq f_1$  nor  $f_1 \leq f_0$ . Since  $K_n$  is a configuration and thereby conflict-free the definition of concurrent events yields  $f_0 \parallel f_1$ . By the contraposition of Lemma 17 follows that with  $\lambda(f_0) = (a_0, \tau_0)$ ,  $\lambda(f_1) = (a_1, \tau_1)$ , the actions  $a_0$  and  $a_1$  are in different components ( $loc(a_0) \neq loc(a_1)$ ) and therefore  $a_0 \neq a_1$  (see definition of  $\Gamma$  in Sect. 2.3). From  $\neg f_0 \# f_1$  and from the definition of  $\#$  we conclude that  $(a_0, a_1) \in I(\hat{\rho}(L_0))$ . Hence  $\lambda(f_0)$  and  $\lambda(f_1)$  are independent at  $\hat{\rho}(L_0)$ .  $\square$

By repeated application of this argument, Claim 18.1 extends to sets of extensions: let  $f_1, \dots, f_m \in K_n$  – with  $\lambda(f_i) = (a_i, \tau_i), i = 1, \dots, m$  – be subsequent extensions of  $\lfloor f_0 \rfloor$  and  $L_0$ , i.e. for all  $1 \leq i \leq m$  let  $L_i := L_{i-1} \uplus \{f_i\}$  with  $f_i$  being an extension of  $L_{i-1}$  and of  $L_{i-1} \uplus \{f_0\}$ . Then for all  $1 \leq i \leq m$ ,  $\lambda(f_0)$  and  $\lambda(f_i)$  are independent at  $\widehat{\rho}(L_{i-1})$ .

Consider the prime configuration  $\lfloor e_n \rfloor$  and let  $K^* = \lfloor e_n \rfloor \setminus \{e_n\}$ . Let  $e \in K_n \setminus \lfloor e_n \rfloor$  be an extension of  $\lfloor e_n \rfloor$ . Let  $K_e := K^* \cap \lfloor e \rfloor$  be their common configuration. The events in  $K^* \setminus K_e$  are subsequent extensions of  $K_e$  and of  $K_e \uplus \{e\}$ . By relating  $e$  to  $f_0$ ,  $K^* \setminus K_e$  to  $\{f_1, \dots, f_m\}$  and assuming w.l.o.g. that  $e_n$  is the last extension, we see that  $\lambda(e)$  and  $\lambda(e_n)$  are independent at  $\widehat{\rho}(K^*)$ . Since  $\lambda(e) \in \text{enabled}(s(K_e))$ , the same action  $\lambda(e)$  is enabled in  $s(K^*)$  by independence to the events of  $K^* \setminus K_e$  at the corresponding configurations.

Furthermore  $\lambda(e_n) \in \text{enabled}(s(K^*))$  because  $e_n$  is a prime trace and  $\text{pref}(e_n) = [\widehat{\rho}(K^*)]_{tr}$ . By the independence of  $\lambda(e)$  and  $\lambda(e_n)$  we may conclude that  $\lambda(e_n) \in \text{enabled}(s(K^* \uplus \{e\}))$ .

Repeating this argument for the remaining events in  $K \setminus \lfloor e_n \rfloor$  in the correct order, we get that  $\lambda(e_n) \in \text{enabled}(s(K_n))$  and therefore we may extend  $K_n$ 's linearization  $(e_1, \dots, e_{n-1})$  by  $e_n$  and get a ta-sequence:  $\lambda(e_1 \dots e_n) \in \text{TaSeq}_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+)$ . This proves the first proposition.

To show the second proposition, we construct the entire equivalence class from equivalence classes of smaller configurations in  $K$ . The critical argument that allows this construction is that concurrent events can be re-ordered in the linearization.

The second proposition holds for the first  $n - 1$  events  $\{e_1, \dots, e_{n-1}\}$  of any linearization  $(e_1, \dots, e_n) \in \text{Lin}(K)$  of  $K$  by inductive assumption:  $\{\lambda(\mathbf{e}') \mid \mathbf{e}' \in \text{Lin}(K_n)\} = [\widehat{\rho}(K_n)]_{tr}$  is a trace.

If  $K = \lfloor e_n \rfloor$  then the proposition is satisfied trivially.

Otherwise there exists a linearization  $(f_1, \dots, f_n) \in \text{Lin}(K)$  with  $e_n \neq f_n$ , with  $L_n := K \setminus \{f_n\}$ , and with the trace  $[\widehat{\rho}(L_n)]_{tr}$  by inductive assumption. Since  $e_n$  and  $f_n$  are extensions of  $K_n \cap L_n$ , and of  $K_n$  and  $L_n$ , respectively, from Claim 18.1 follows that  $\lambda(e_n)$  and  $\lambda(f_n)$  are independent at  $\widehat{\rho}(K \setminus \{e_n, f_n\})$ . Consequently for each ta-sequence  $\underline{\rho} \in [\widehat{\rho}((K \setminus \{e_n, f_n\}))]_{tr}$ , the ta-sequences  $\underline{\rho} \lambda(e_n) \lambda(f_n)$  and  $\underline{\rho} \lambda(f_n) \lambda(e_n)$  are equivalent. Repeating this argument for any pair  $e_n, f_n$  of causally maximal events of  $K$ , we can construct the complete equivalence class  $\{\lambda(\mathbf{e}) \mid \mathbf{e} \in \text{Lin}(K)\} = [\widehat{\rho}(K)]_{tr}$ , which is a trace.  $\square$

We may therefore write  $\text{state}_{R,l}^{\circ, \mathbf{g}}(K)$  for  $\text{state}_{R,l}^{\circ, \mathbf{g}}(\lambda(\mathbf{e}))$ , with  $\mathbf{e} \in \text{Lin}(K)$ . We will also say that  $ES^{\mathbf{g}}(\mathcal{A}^+)$  is the *unfolding* of  $\mathcal{A}$  and write  $Unf(\mathcal{A})$  for it.

## 2.10 Summary of Networks of Timed Automata

We want to summarize what has been achieved in the previous sections. We defined networks of message passing timed automata (MTA) in Sect. 2.3 as an extension of timed

automata: The components of a MTA are timed automata that synchronize by exchanging messages. We considered a special class of these networks that satisfy the structural constraints of netcharts. We define their standard semantics (global, concrete semantics) and showed in Sect. 2.6 that the idea of Bengtsson et al. [BJLY98], to desynchronize the progress of time component-wise is applicable to MTA as well. The intermediate steps of introducing reference clocks and time-stamping messages were necessary to show this result (Sect. 2.5).

We then showed that the region abstraction for timed automata can be extended to MTA and that in the case of  $B$ -bounded MTA, the region automaton finitely represents the state space of the MTA (see Sect. 2.8). We then desynchronized the progress of time in the region automaton of an MTA and defined local-region semantics in Sect. 2.9. In Sect. 2.9.1 we showed that in local-region semantics, concurrently enabled actions are independent which allowed us to define an event structure that represents exactly the runs of a MTA in local-region semantics by ordering concurrent events partially, see Sect. 2.9.2.

In the next section, we will prove that the event structure also represents the runs of a MTA in global, concrete semantics. The subsequent Sect. 4 contains the proof that this event structure has a complete, finite prefix. This implies that the event structure is a finite representation of the state space of a MTA where concurrent events are partially ordered.

### 3 Local Region Automata are equivalent to Timed-Stamped Networks of Timed Automata

This section is dedicated to the proof that for any MTA  $\mathcal{A}$  with corresponding MTA<sup>+</sup>  $\mathcal{A}^+$ , its local region automaton  $R_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$  is equivalent to its transition system  $TS^\circ(\mathcal{A})$ . Though, this equivalence does not apply to any reachable region but only those, where the same amount of time has passed in each component. This restriction is natural by the global character of time in the original semantics of MTA<sup>+</sup> where synchronicity is necessary for reachable states.

As for states in local clock semantics, we introduce synchronized and synchronizable regions. A decomposed region  $s = ((w_1, \alpha_1), \dots, (w_P, \alpha_P)), \beta$  is *synchronized* if for all  $p, q \in \mathcal{P}$ ,  $\alpha_p(g_p) = \alpha_q(g_q)$  holds. A decomposed region  $s$  can *catch up* to a decomposed region  $\vec{s}$  if there exists decomposed regions  $s_0, s_1, \dots, s_P$ ,  $s_0 = s$ ,  $s_P = \vec{s}$ , such that  $s_{i-1} \xrightarrow{\delta} s_i$  or  $s_{i-1} = s_i$  for all  $i \in \mathcal{P}$ . The region  $s$  is *synchronizable* if  $s$  can catch up to the synchronized region  $\vec{s}$  with  $\vec{\alpha}_p(g_p) = \max\{\alpha_q(g_q) \mid q \in \mathcal{P}\}$  for all  $p \in \mathcal{P}$ .

We now show that the events of a configuration can be linearly ordered such that they correspond to a ta-sequence in region-stamped semantics. Furthermore, the resulting states are separated by some local time steps only: in the state reached by the configuration in local-region semantics, all components can catch up to the state that is reached by region-stamped semantics.

**Proposition 19:** *Let  $K \subseteq E$ .*

1. *There exists  $(e_1, \dots, e_n) \in \text{Lin}(K)$  such that and for all  $1 \leq i < j \leq n$  with  $\lambda(e_i) = (a_i, \tau_i)$ ,  $\lambda(e_j) = (a_j, \tau_j)$ ,  $\tau_i \leq \tau_j$  holds.*
2. *If  $\text{state}_{R,l}^{\circ, \mathbf{g}}(K) = s = ((w_1, \alpha_1), \dots, (w_P, \alpha_P), \beta)$  is synchronizable and  $\mathbf{e} \in \text{Lin}(K)$  is timely ordered as in claim 1, then*
  - $\lambda(\mathbf{e}) \in \text{TaSeq}_R^{\circ, \mathbf{g}}(\mathcal{A}^+)$ ,
  - $\text{state}_R^{\circ, \mathbf{g}}(\lambda(\mathbf{e})) = \bar{s} = ((w_1, \dots, w_P), \bar{\alpha}, \beta)$  with  $\pi_{\mathcal{P}}(\bar{s}) = \vec{s}$  and  $s$  can catch up to  $\vec{s}$ , and
  - $\text{enabled}_R^{\circ, \mathbf{g}}(\bar{s}) = \text{enabled}_{R,l}^{\circ, \mathbf{g}}(\vec{s})$ . \*

*Proof.* We sketch the proof only: For claim 1. By induction on  $|K| = n$  it is sufficient to show that any two directly causally ordered events are also ordered timely by their reference clocks: for  $e, e' \in K$  if  $e \leq e'$  and  $e$  is a direct predecessor of  $e'$  (there exists no  $e''$  with  $e' \leq e'' \leq e$ ) with  $\lambda(e') = (a', \tau')$ ,  $\text{esLab}(e) = (a, \tau)$ , in  $s' = \text{state}_{R,l}^{\circ, \mathbf{g}}(e')$  and  $s = \text{state}_{R,l}^{\circ, \mathbf{g}}(e)$  holds:  $\tau \leq \tau'$ . This property follows from the time-progress in the same component if  $\text{loc}(a) = \text{loc}(a')$  and from the time-stamping if  $a$  and  $a'$  are corresponding communicating actions.

By reordering concurrent events, we obtain a linearization that is monotonically increasing in the action's time-stamps and that obeys causality.

Claim 2 is proven by induction on  $|K| = n$  as well. The base case  $n = 0$  is trivial and for a configuration  $K$  and a corresponding local region ta-sequence  $\rho = \rho' (a, \tau) = \lambda(\mathbf{e})$  ( $\mathbf{e} \in L(K)$  satisfying claim 1) with  $n > 0$  consider the reached states  $\bar{s}' = state_R^{\circ, \mathbf{g}}(\rho')$ ,  $s' = state_{R,l}^{\circ, \mathbf{g}}(\rho')$  and the catch-up state  $\vec{s}'$ .

The region-stamp  $\tau$  of  $a$  is larger than any region-stamp in  $\vec{s}'$  and  $\bar{s}'$  and  $(a, \tau) \in enabled_{R,l}^{\circ, \mathbf{g}}(\vec{s}') = enabled_R^{\circ, \mathbf{g}}(\bar{s}') = enabled_{R,l}^{\circ, \mathbf{g}}(s')$  by induction hypothesis. Thus we can let time pass in either region to reach the reference time  $\tau$ .

In  $\bar{s}$  and  $\vec{s}$ , invariants are evaluated equivalently, allowing the same amount of time to pass. Furthermore guards are evaluated equivalently. In  $s$ ,  $\vec{s}$  and  $\bar{s}$  the number of messages in each buffer is equal and less than  $B$ , thus a time-stamped action  $(b, \tau_b)$  is enabled in  $\bar{s}$  iff it is enabled in  $\vec{s}$ .  $\square$

Conversely, every region-stamped ta-sequence is a local region ta-sequence and thus has a corresponding configuration; following the same ta-sequence, the state reached by region-stamped semantics and the state reached by local region semantics are separated by synchronizing local time steps.

**Proposition 20:**  $TaSeq_R^{\circ, \mathbf{g}}(\mathcal{A}^+) \subseteq TaSeq_{R,l}^{\circ, \mathbf{g}}(\mathcal{A}^+)$ .  $\star$

*Proof.* Proof sketch: every global time step can be simulated by all components together (subsequently), subsequent time steps of one component can be composed to a single larger time step without reordering discrete steps (by independence of time steps).  $\square$

**Proposition 21:** For all ta-sequences  $\rho \in TaSeq_R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  of a  $MTA^+$  in region stamped semantics with  $state_R^{\circ, \mathbf{g}}(\rho) = \bar{s}$ ,  $state_{R,l}^{\circ, \mathbf{g}}(\rho) = s$ ,  $s$  can catch up to  $\pi_{\mathcal{P}}(\bar{s})$ .  $\star$

*Proof (by induction on  $|\rho|$ ).* Proof sketch: trivially true for  $|\rho| = 0$ . For longer ta-sequences  $\rho = \rho'(a, \tau)$  where  $\rho'$  yields local-region  $s'$ , global region  $\bar{s}'$  and catch-up region  $\pi_{\mathcal{P}}(\bar{s}') = \vec{s}'$ , the component  $p = loc(a)$  of the last firing action can make a time-step from  $s'$  to reach reference time  $\tau$  and fire  $a$ .

Since region-stamped ta-sequences are timely monotonically increasing,  $\tau = \bar{\alpha}(g_p)$  is greater than or equal to all reference clocks in  $\vec{s}'$ . Furthermore, all components can do the time-step from  $\bar{\alpha}'$  to  $\bar{\alpha}$  with  $\bar{\alpha}(g_p) = \tau$  by definition of ta-sequences and  $state_R^{\circ, \mathbf{g}}$ . Since in  $s'$  the components  $q \in \mathcal{P}$ ,  $q \neq p$ , can catch up to  $\vec{s}'$ , and  $\alpha_q = \alpha'_q$ , they can catch up to  $\vec{s} = \pi_{\mathcal{P}}(\bar{s})$  as well. Thus  $\bar{\alpha}(g_q) = \bar{\alpha}(g_q) = \tau$  holds. By the definition of ta-sequences, component  $p$  catches up to global time  $\tau$  before firing  $a$ . Hence in  $s$  all components can catch up in time to reach  $\vec{s}$ .  $\square$

The results of Propositions 19, 20, and 21 yield a first central result of this Thesis.

**Theorem 22.** Let  $s$  be a stamped region of a  $MTA^+$   $\mathcal{A}^+$ .

$s$  is reachable by region stamped semantics via ta-sequence  $\rho$  iff there exists a synchronizable, decomposed region  $\tilde{s}$  that is reachable by local region semantics via  $\tilde{\rho} \in [\rho]_{tr}$  such that  $\pi_{\mathcal{P}}(s)$  can be reached from  $\tilde{s}$  by local time steps  $\xrightarrow{\delta}_p$  only.  $\star$

*Proof.* Follows from the preceding propositions.  $\square$

From this theorem follows the reachability equivalence of synchronized states between local region semantics and the MTA's underlying transition system under the region abstraction.

**Corollary 23:** *Let  $\mathcal{A}$  be a MTA and  $\mathcal{A}^+$  the corresponding MTA<sup>+</sup> with reference clocks  $\mathbf{g}$ .*

*For every ta-sequence  $\hat{\rho} \in \text{TaSeq}^\circ(\mathcal{A})$  of  $TS(\mathcal{A})$  exists a ta-sequence  $\hat{\rho}_l \in \text{TaSeq}_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$  of  $R_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$  such that the following property holds:*

$$\begin{aligned} \hat{\rho}_l &= \text{abstr}(\hat{\rho}) \text{ and } \text{state}^\circ(\hat{\rho}) = (\mathbf{w}, \sigma, \beta) \\ &\text{and } \text{state}_{R,l}^{\circ, \mathbf{g}}(\hat{\rho}_l) \text{ can catch up to the region } ((w_1, \alpha_1), \dots, (w_P, \alpha_P), \beta_l) \text{ where} \\ \mathbf{w} &= (w_1, \dots, w_P) \wedge \forall p \in \mathcal{P} : \sigma|_p \in \Pi_{C_p}(\alpha_p) \\ &\wedge \forall p, q \in \mathcal{P} \forall m \in M(p, q) : |\beta_l(p, q, m)| = \beta(p, q, m) \end{aligned} \quad (4)$$

*Conversely, for every ta-sequence  $\hat{\rho}_l^* \in \text{TaSeq}_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$  of  $R_l^{\circ, \mathbf{g}}(\mathcal{A}^+)$  exists an equivalent ta-sequence  $\hat{\rho}_l \in [\hat{\rho}_l^*]_{tr}$  and a ta-sequence  $\hat{\rho} \in \text{TaSeq}^\circ(\mathcal{A})$  of  $TS(\mathcal{A})$  such that property (4) holds for  $\hat{\rho}$  and  $\hat{\rho}_l$ .*  $\star$

We thereby have shown that the event structure  $Unf(\mathcal{A})$  equivalently expresses the state space of a MTA  $\mathcal{A}$ .

## 4 Event Structures of Local Region Automata have a Finite, Complete Prefix

In this section we are going to postulate and prove the main result of this Thesis, that the state space of a  $B$ -bounded network of message passing timed automata can be represented as a complete, finite prefix of an unfolding.

**Theorem 24.** *Let  $\mathcal{A}^+$  be a progressing MTA<sup>+</sup> with reference clocks  $\mathbf{g}$ . The underlying region automaton  $R_l^{\mathbf{g}}(\mathcal{A}^+)$  can be represented in terms of a finite complete prefix of its unfolding.* \*

Although we presume that any  $B$ -bounded MTA has a complete, finite prefix, we can currently substantiate the result for a restricted, but practically relevant class only, which we call *progressing* MTA. We will prove the existence of such a prefix for this class and we will analyze some properties of the unfolding that are necessary to calculate the prefix algorithmically. However, we will not give an algorithmic solution for the problem, which is thereby left as future work.

To achieve this result, two major steps need to be taken. Firstly, we have to show that despite the presence of unbounded reference clock variables, the region equivalence relation is of finite index and hence the region automaton is finitely representable. This is done in Sect. 4.1 and involves rather technical arguments that transfer a result from [Min99b] to our current setting. Secondly, we show that a progressing MTA<sup>+</sup>'s event structure has a complete prefix of finite size. Here, the major argument exploits that the state space is finitely representable and that the event structure equivalently represents the state space.

### 4.1 Region Equivalence and Reference Clocks

The reader who is willing to believe that the presence of reference clocks does not affect the finite representability of the region automaton may skip the technical proofs of this section and continue at Sect. 4.2. The main argument is that for region equivalence, the presence and the value of a reference clock does not affect a timed automaton's behaviour.

The approach for a finite representation and the corresponding proof sketches have been given first by Minea in [Min99b]. However, for the final proof that the finite representation is given due to a region equivalence relation, we propose a different approach which is more revealing about the nature of reference clocks.

The proof is given for a timed automaton with a single reference clock. The finite representation is then applicable to MTA<sup>+</sup> in local region semantics by applying it component wise.

**Transforming Clock Assignments** Let  $A = (Q, \Gamma, C, \longrightarrow, inv, w^0)$  be a timed automaton. Let  $g \notin C$  be the reference clock of  $A + g$ . We define the *clock coordinate transfor-*

ation  $\Delta_g : \Sigma(C^+) \rightarrow \Sigma(C^+)$ :  $\Delta_g(\sigma) = \tilde{\sigma}$  where

$$\tilde{\sigma}(x) =_{def} \begin{cases} \sigma(x), & \text{if } x = g \\ \sigma(g) - \sigma(x), & \text{if } x \neq g \end{cases}$$

$\Delta_g$  is a bijection. Furthermore,  $\Delta_g(\Delta_g(\sigma)) = \sigma$ . In the following, we will use tilded symbols ( $\tilde{\sigma}$ ) when referring to objects in the transformed representation.

In the new coordinates, we define a time-pass to advance  $g$  only, and a clock-reset to set the clocks to the current value of  $g$ :

- $(\tilde{\sigma} +_{\Delta} d)(x) = \tilde{\sigma}(x) + d$  if  $x = g$  and  $(\tilde{\sigma} +_{\Delta} d)(x) = \tilde{\sigma}(x)$  otherwise.
- for  $R \subseteq C$ :  $\tilde{\sigma}[R \mapsto 0]_{\Delta}(x) = \tilde{\sigma}(g)$  if  $x \in R$  and  $\tilde{\sigma}[R \mapsto 0]_{\Delta}(x) = \tilde{\sigma}(x)$  otherwise.

**Proposition 25:** *Let  $\sigma \in \Sigma(C^+)$  and  $R \subseteq C$ . Then*

1.  $\Delta_g(\sigma)[R \mapsto 0]_{\Delta} = \Delta_g(\sigma[R \mapsto 0])$ .
2.  $(\sigma +_{\Delta} d) = \Delta_g(\sigma + d)$  \*

*Proof.* Follows from applying the definition of  $\Delta_g$  to the reset and progress of time.  $\square$

By  $\bowtie$  we denote an arbitrary but fixed element from  $\{\leq, <, =, >, \geq\}$ . For a fixed  $\bowtie$ , let  $\bowtie^{-1}$  denote the inverse relation, e.g.  $<^{-1} =_{def} >$ .

**Proposition 26:** *Let  $k \in \mathbb{Z}$  and  $x, y \in C$ ,  $\sigma \in \Sigma(C^+)$  and  $\bowtie \in \{<, \leq, =, \geq, >\}$ .*

1.  $\sigma \models x = k$  iff  $\Delta_g(\sigma) \models x = g - k$ .
2.  $\sigma \models k < x < k + 1$  iff  $\Delta_g(\sigma) \models x = g - k > x > g - (k + 1)$ .
3.  $\sigma \models x > k$  iff  $\Delta_g(\sigma) \models x < g - k$ .
4.  $\sigma \models x - y \bowtie k$  iff  $\Delta_g(\sigma) \models y - x \bowtie k$ .
5.  $\sigma \models \text{fract}(x) = 0$  iff  $\Delta_g(\sigma) \models \text{fract}(x) = \text{fract}(g)$
6.  $\sigma \models \text{fract}(x) \bowtie \text{fract}(y)$  iff

$$\begin{aligned} \Delta_g(\sigma) \models & ( (\text{fract}(g) \leq \text{fract}(g - x) \wedge \text{fract}(g) \leq \text{fract}(g - y)) \\ & \vee (\text{fract}(g) \geq \text{fract}(g - x) \wedge \text{fract}(g) \geq \text{fract}(g - y)) ) \\ & \wedge \text{fract}(x) \bowtie^{-1} \text{fract}(y) \\ & \vee (\text{fract}(g) \leq \text{fract}(g - x) \wedge \text{fract}(g) \geq \text{fract}(g - y)) \\ & \wedge \text{fract}(x) - 1 \bowtie^{-1} \text{fract}(y) \\ & \vee (\text{fract}(g) \geq \text{fract}(g - x) \wedge \text{fract}(g) \leq \text{fract}(g - y)) \\ & \wedge \text{fract}(x) \bowtie^{-1} \text{fract}(y) - 1 \end{aligned}$$
\*

*Proof.* The soundness of the transformation-rules 1 to 5 follows from transforming the left-hand sides into the right-hand sides by applying the definition of  $\Delta_g$ . The last transformation 6 requires case distinctions because the transformation may affect the size of the fractional part of a clock variable.  $\square$

Let  $\psi$  be any of the formulas on the left-hand side of Proposition 26. Then by  $\Delta_g(\psi)$  denote the corresponding formula on the right-hand side.

Remember from the definition of region equivalence (see Sect. 2.7), that for each region  $\alpha \in R(C^+)$  exists a conjunction  $\varphi_\alpha$  of constraints of the form  $x = k$ ,  $k < x < k + 1$ ,  $x > k_x$  (one per  $x \in C^+$ ) and  $fract(x) \bowtie fract(y)$  (one per pair  $(x, y) \in C^+ \times C^+$ ) such that  $\sigma \in \alpha$  iff  $\sigma \models \varphi_\alpha$  holds.

By  $\Delta_g(\varphi)$  denote the conjunction where each constraint  $\psi$  is replaced by  $\Delta_g(\psi)$ , furthermore  $\Delta_g(\alpha) =_{def} \{\Delta_g(\sigma) \mid \sigma \in \alpha\}$ . The definition of  $\Delta_g(\alpha)$  is sound as the following proposition shows.

**Proposition 27:**  $\sigma \models \varphi_\alpha$  iff  $\Delta_g(\sigma) \models \Delta_g(\varphi_\alpha)$ . \*

*Proof.*  $\sigma \models \varphi_\alpha$  iff  $\forall \psi \in \varphi_\alpha : \sigma \models \psi$  (by definition) iff  $\forall \psi \in \varphi_\alpha : \Delta_g \sigma \models \Delta_g \psi$  (by Prop. 26) iff  $\Delta_g(\sigma) \models \Delta_g(\varphi_\alpha)$  (by definition).  $\square$

By the definition of  $\Delta_g$  for formulas  $\psi$ , the region operators  $.[R \mapsto 0]$  and  $succ_r(\cdot)$  that transform the symbolic representation of a clock region can be translated into the new representation. We define the new operators:  $.[R \mapsto 0]_\Delta$  and  $succ_{r,\Delta}(\cdot)$  in the new representation.

**Proposition 28:** Let  $\alpha \in R(C^+)$  and  $R \subseteq C$ . Then

1.  $\Delta_g(\alpha)[R \mapsto 0]_\Delta = \Delta_g(\alpha[R \mapsto 0])$ .

2.  $succ_{r,\Delta}(\Delta_g(\alpha)) = \Delta_g(succ_r(\alpha))$  \*

*Proof.* We sketch the proof idea: The original operators ‘reset’ and ‘successor region’ can be defined in terms of clock constraints of a region by replacing clock constraints in a well-defined way [AD94]. Through the one-to-one correspondence of original and transformed clock constraints by Proposition 26 and the corresponding re-definition of ‘reset’ and ‘successor region’ for the transformed regions, the equations hold.  $\square$

By the last proof we established that the clock regions of the region automaton can be represented in a different way through the  $\Delta_g$  operator and that the dynamics can entirely be defined in the new representation. What remains to be shown is that there are only finitely many regions that yield different behaviour in the region automaton. We do so by defining a new equivalence relation on transformed clock valuations. We then show that this equivalence relation is a region equivalence relation and that it has a finite index.

The idea for the region equivalence relation in the transformed representation is based on [Min99b]. Let  $c_{min}$  and  $c_{max}$  be the minimal and the maximal constant occurring in  $A$ .

**Definition 21.** Two clock assignments  $\tilde{\sigma}, \tilde{\sigma}' \in \Sigma(C^+)$  are called *region-equivalent* in the transformed representation (denoted by  $\tilde{\sigma} \sim_{r,\Delta} \tilde{\sigma}'$ ) if for any clock variables  $x, y \in C^+$ , one of the following conditions holds:

1.  $c_{min} \leq \lfloor \tilde{\sigma}(x) - \tilde{\sigma}(y) \rfloor = \lfloor \tilde{\sigma}'(x) - \tilde{\sigma}'(y) \rfloor \leq c_{max}$ , and  $\tilde{\sigma}(x) - \tilde{\sigma}(y) \in \mathbb{Z}$  iff  $\tilde{\sigma}'(x) - \tilde{\sigma}'(y) \in \mathbb{Z}$
2.  $\lfloor \tilde{\sigma}(x) - \tilde{\sigma}(y) \rfloor < c_{min}$  and  $\lfloor \tilde{\sigma}'(x) - \tilde{\sigma}'(y) \rfloor < c_{min}$
3.  $\lfloor \tilde{\sigma}(x) - \tilde{\sigma}(y) \rfloor > c_{max}$  and  $\lfloor \tilde{\sigma}'(x) - \tilde{\sigma}'(y) \rfloor > c_{max}$  \*

**Proposition 29:** Let  $A + g$  be a timed automaton with reference clock  $g$  and  $C^+ = C \uplus \{g\}$ . Let  $\sigma, \sigma' \in \Sigma(C^+)$  and  $\tilde{\sigma} = \Delta_g(\sigma), \tilde{\sigma}' = \Delta_g(\sigma')$ .

1.  $\tilde{\sigma} \sim_{r,\Delta} \tilde{\sigma}'$  implies  $\sigma|_C \sim_r \sigma'|_C$ , and
2.  $\sigma|_C \sim_r \sigma'|_C$  implies  $\tilde{\sigma} \sim_{r,\Delta} \tilde{\sigma}'$ . \*

The proposition above states that the value of reference clock  $g$  in a clock valuation  $\sigma$  does not influence how a timed automaton behaves, which is actually not surprising at all. By this property we see that the technicality of a reference clock does no harm to the region equivalence which is necessary for a finite representation of a timed automaton's state space.

The proof shows that the conditions of either region equivalence imply the the conditions of the other region equivalence and that the actual value of reference clock  $g$  is of no importance.

*Proof.* Proof sketch: for claim 1, by transforming the conditions of Definition 21 back into the original representation we see that if  $\tilde{\sigma}$  and  $\tilde{\sigma}'$  are equivalent by  $\sim_{r,\Delta}$ , the equivalence conditions of  $\sim_r$  for  $\sigma$  and  $\sigma'$  are satisfied. Since the transformation of the conditions is done in equivalent transformation steps, claim 2 holds as well.  $\square$

Let  $\tilde{\sigma} \in \tilde{\alpha}$  and  $\tilde{\sigma}' \in \tilde{\alpha}'$  be two clock valuations in the transformed representation. In contrast to the clock regions of timed automata without reference clocks, it is possible that  $\tilde{\alpha} \neq \tilde{\alpha}'$  but  $\tilde{\sigma} \sim_{r,\Delta} \tilde{\sigma}'$  as the clock regions may disagree on their reference clock:  $\tilde{\alpha}(g) \neq \tilde{\alpha}'(g)$ . In order to compare clock regions such that they are considered equivalent iff their clock valuations are region equivalent, we have to define an equivalence relation on clock regions itself.

By overloading notation, let  $\tilde{\alpha} \sim_{r,\Delta} \tilde{\alpha}'$  iff there exist  $\tilde{\sigma} \in \tilde{\alpha}$  and  $\tilde{\sigma}' \in \tilde{\alpha}'$  such that  $\tilde{\sigma} \sim_{r,\Delta} \tilde{\sigma}'$ . From  $\sim_{r,\Delta}$  being an equivalence relation follows that

$$\tilde{\alpha} \sim_{r,\Delta} \tilde{\alpha}' \text{ implies } \forall \tilde{\sigma} \in \tilde{\alpha}, \tilde{\sigma}' \in \tilde{\alpha}' : \tilde{\sigma} \sim_{r,\Delta} \tilde{\sigma}'.$$

The preceding results in Propositions 29 and 27 allow us to conclude that for any two regions  $\alpha, \alpha' \in R(C^+)$  holds:  $\Pi_C(\alpha) = \Pi_C(\alpha')$  iff  $\Delta_g(\alpha) \sim_{r,\Delta} \Delta_g(\alpha')$ . Hence we transfer the notion of equivalent regions to the non-transformed notion:

$$\alpha \sim_{r,\Delta} \alpha' \text{ :} \Leftrightarrow \Pi_C(\alpha) = \Pi_C(\alpha').$$

We thereby obtain an equivalence relation on clock regions *with* reference clocks that renders clock regions equivalent if they are not equal due to the value of reference clocks only.

**Proposition 30:**  $\sim_{r,\Delta}$  over clock valuations has finite index. \*

*Proof.* From the definition of  $\sim_{r,\Delta}$  one can see that there exist only finitely many different equivalence classes: since  $C^+$  is finite and since there are only finitely many different integers between  $c_{min}$  and  $c_{max}$  and either the difference of two clocks is an integer in both clock valuations or it is not, there are only finitely many clock values that yield a different result on the conditions of Definition 21. □

The finiteness-result extends to  $\sim_{r,\Delta}$  on clock regions as well because the definitions base by equivalence on  $\sim_{r,\Delta}$  over clock valuations.

## 4.2 State Equivalence and Finite Representability of Region-Stamped Automata

In the previous section we have defined an equivalently expressive equivalence relation  $\sim_{r,\Delta}$  on clock regions of timed automata with reference clocks such that two region are said to be equivalent iff they evaluate clock formulas equivalently and the region operations yield equivalent results regardless of the value of the reference clock. We have shown furthermore that this equivalence relation has a finite index.

This implies that the state space of a timed automaton with a reference clock can be finitely represented in terms of a region automaton. We now want to extend this result to networks of message passing timed automata (with reference clocks). The difficulty here is that the buffer-state functions map to multi-sets of region stamps of which there are infinitely many. We prove that this does no harm to the finite representability. We need this result to prove the existence of a complete, finite prefix of an unfolding of a MTA in the next section.

Our approach for the proofs is as follows: In Proposition 7 we have shown that in the region automaton  $R(\mathcal{A})$  of an MTA  $\mathcal{A}$  without reference clocks, the set of reachable regions is finite. By adding reference clocks  $\mathbf{g} = (g_1, \dots, g_p)$  to  $\mathcal{A}$  we get the MTA<sup>+</sup>  $\mathcal{A}^+$ , and we can compute the region-stamped automaton  $R^{\mathbf{g}}(\mathcal{A}^+)$  which is bisimilar to  $R(\mathcal{A})$  by the bisimulation relation *sim* from (1), see Proposition 9. The bisimulation relation *sim* and the alternative region equivalence relation  $\sim_{r,\Delta}$  are “compatible”; from their combination we can deduce that the state space of  $R^{\mathbf{g}}(\mathcal{A}^+)$  is finitely representable.

We extend the notion of equivalent clock regions to stamped regions of  $RS^{\mathbf{g}}(\mathcal{A}^+)$ . It is insufficient to compare clock regions  $\alpha, \alpha' \in R(C \uplus \{g_1, \dots, g_p\})$  through equality: by the presence of a reference clock  $g_1$ , there are infinitely many different representations for the same set of constraints on the “normal” clock variables  $C$  of the system, i.e.  $\Pi_C(\alpha) = \Pi_C(\alpha')$  and  $\alpha(g_1) \neq \alpha'(g_1)$ . However the results of Proposition 29 and of the preceding Sect. 4.1 show that  $\alpha \sim_{r,\Delta} \alpha'$  is necessary and sufficient to yield equivalent behaviour of a single timed automaton.

Observe first that in the case of several reference clocks,  $\sim_{r,\Delta}$  is well-defined if all reference clocks have the same value. This restriction on the clock regions poses no problem as it is satisfied by all reachable regions of  $R^{\mathbf{g}}(\mathcal{A}^+)$ , see Sect. 2.8.

A region  $s = (\mathbf{w}, \alpha, \beta) \in RS^{\mathbf{g}}(\mathcal{A}^+)$  is called *consistent* if for all  $p, q \in \mathcal{P}, m \in M(p, q)$  and for all  $\tau \in \beta(p, q, m)$ ,  $\tau \leq \alpha(g_p)$ , i.e. the buffers contain no messages that cannot have been sent yet. Note that all reachable regions of  $R^{\circ\mathbf{g}}(\mathcal{A}^+)$  are consistent.

We observe that an action  $q?p(m)$  is enabled in a consistent region  $s = (\mathbf{w}, \alpha, \beta)$  iff  $w_q \xrightarrow{\varphi, q?p(m), R} w'_q$ ,  $\alpha \models \varphi$ ,  $\alpha[R \mapsto 0] \models \text{inv}(w'_q)$ , and  $|\beta(p, q, m)| > 0$ . This allows us to define an equivalence relation on consistent regions  $s_1 = (\mathbf{w}_1, \alpha_1, \beta_1), s_2 = (\mathbf{w}_2, \alpha_2, \beta_2) \in RS^{\mathbf{g}}(\mathcal{A}^+)$ :  $s_1 \simeq^{\mathbf{g}} s_2$  iff  $s_1$  and  $s_2$  are consistent and

$$\begin{aligned} \mathbf{w}_1 &= \mathbf{w}_2 \wedge \alpha_1 \sim_{r,\Delta} \alpha_2 \\ \wedge \forall p, q \in \mathcal{P} \forall m \in M(p, q) : |\beta_1(p, q, m)| &= |\beta_2(p, q, m)|. \end{aligned}$$

By  $[s]_{\simeq}$  we denote the equivalence class of  $s$  by  $\simeq^{\mathbf{g}}$ .

We can show that equivalent stamped regions of  $R^{\mathbf{g}}(\mathcal{A}^+)$  correspond by the bisimulation relation to the same region of  $R(\mathcal{A})$ .

**Lemma 31:** *Let  $\text{sim}$  be the bisimulation predicate from (1) in Proposition 9 and let  $t_1, t_2 \in RS(\mathcal{A})$  be regions and  $s_1, s_2 \in RS^{\mathbf{g}}(\mathcal{A}^+)$  be stamped regions such that  $\text{sim}(t_1, s_1)$  and  $\text{sim}(t_2, s_2)$ . Then  $s_1 \simeq^{\mathbf{g}} s_2$  iff  $t_1 = t_2$ . \**

*Proof.* Taking the definition of  $\text{sim}$  from Proposition 9, the equivalence on the discrete states and the buffers follows. Since  $\text{sim}$  holds for  $s_1$  and  $s_2$  both regions are consistent which implies that  $s_1 \simeq^{\mathbf{g}} s_2$  has a well-defined value. This value is the same as  $t_1 = t_2$  by Proposition 29. □

There are infinitely many reachable regions in  $R^{\mathbf{g}}(\mathcal{A}^+)$  that differ in the region stamps in the buffers. But from the finite number of reachable regions of  $R(\mathcal{A})$ , the finiteness of  $\sim_{r,\Delta}$  and the above Lemma 31 follows that there are only finitely many different regions in  $R^{\mathbf{g}}(\mathcal{A}^+)$  that yield a different behaviour; we show why that is the case.

For a ta-sequence  $\rho \in \text{TaSeq}_R^{\circ\mathbf{g}}(\mathcal{A}^+)$ , let  $\uparrow\rho$  denote the set of all ta-sequences  $\bar{\rho}$  that are an *extension* of  $\rho$ :  $\rho \bar{\rho} \in \text{TaSeq}_R^{\circ\mathbf{g}}(\mathcal{A}^+)$  – in analogy to the extensions of configurations. Now consider ta-sequences  $\rho_1, \rho_2 \in \text{TaSeq}_R^{\circ\mathbf{g}}(\mathcal{A}^+)$  with  $\text{state}_R^{\circ\mathbf{g}}(\rho_i) \simeq^{\mathbf{g}} \text{state}_R^{\circ\mathbf{g}}(\rho_i)$ .

By Corollary 10, any two equivalent regions  $s_1, s_2 \in RS^{\mathbf{g}}(\mathcal{A}^+)$ ,  $s_1 \simeq^{\mathbf{g}} s_2$  are bisimilar to the same set of regions in  $R(\mathcal{A})$ . Additionally,  $\text{sim}$  covers *all* reachable regions in  $R(\mathcal{A})$  and in  $R^{\mathbf{g}}(\mathcal{A}^+)$ . For every step  $s_1 \xrightarrow{\delta, a} s'_1$  done in  $R^{\mathbf{g}}(\mathcal{A}^+)$  exists a step  $t \xrightarrow{\delta, a} t'$  with  $\text{sim}(t, s_1)$  and  $\text{sim}(t', s'_1)$ . By the bisimilarity  $\text{sim}(t, s_2)$  there exists also a step  $s_1 \xrightarrow{\delta, a} s'_2$  in  $R^{\mathbf{g}}(\mathcal{A}^+)$  with  $\text{sim}(t', s'_2)$ . From Lemma 31 follows that  $s'_1 \simeq^{\mathbf{g}} s'_2$  holds.

**Lemma 32:** *Let  $\mathcal{A}^+$  be a  $\text{MTA}^+$  with reference clocks  $\mathbf{g}$ .*

1. *The equivalence relation  $\simeq^{\mathbf{g}}$  on regions  $RS^{\mathbf{g}}(\mathcal{A}^+)$  is a bisimulation relation from  $R^{\mathbf{g}}(\mathcal{A}^+)$  to itself.*

2. Let  $\rho_1, \rho_2 \in \text{TaSeq}_R^{\circ, \mathbf{g}}(\mathcal{A}^+)$  with  $\text{state}_R^{\circ, \mathbf{g}}(\rho_1) \simeq^{\mathbf{g}} \text{state}_R^{\circ, \mathbf{g}}(\rho_2)$ . Then for any  $\bar{\rho}_1 \in \uparrow \rho_1$  exists  $\bar{\rho}_2 \in \uparrow \rho_2$ ,  $|\bar{\rho}_1| = |\bar{\rho}_2|$  yielding equivalent regions:

$$\text{state}_R^{\circ, \mathbf{g}}(\rho_1 \bar{\rho}_1) \simeq^{\mathbf{g}} \text{state}_R^{\circ, \mathbf{g}}(\rho_2 \bar{\rho}_2).$$

*Proof.* The proof for the first proposition has just been given above. The proof for 2. follows by induction on the length of the extensions.  $\square$

Because  $\simeq^{\mathbf{g}}$  characterizes a non-trivial bisimulation relation from  $R^{\mathbf{g}}(\mathcal{A}^+)$  to itself, where bisimilar regions represent the same regions in  $R(\mathcal{A})$ , we may omit redundant regions (and traces). Let  $[R^{\mathbf{g}}(\mathcal{A}^+)]_{\simeq} = ([RS^{\mathbf{g}}(\mathcal{A}^+)]_{\simeq}, \rightarrow_{\simeq}, [s^0]_{\simeq})$  be the quotient transition system of  $R^{\mathbf{g}}(\mathcal{A}^+)$  under  $\simeq^{\mathbf{g}}$ , where

$$[s]_{\simeq} \xrightarrow{\delta, a}_{\simeq} [s']_{\simeq} \text{ iff } \exists s_1 \in [s]_{\simeq}, s'_1 \in [s']_{\simeq} : s_1 \xrightarrow{\delta, a} s'_1.$$

$[R^{\mathbf{g}}(\mathcal{A}^+)]_{\simeq}$  is isomorphic to  $R(\mathcal{A})$  by *sim*, and bisimilar to  $R^{\mathbf{g}}(\mathcal{A}^+)$ . The finiteness of  $\simeq^{\mathbf{g}}$  implies the finiteness of the quotient transition system wrt reachable regions. Thus in the state space of  $R^{\mathbf{g}}(\mathcal{A}^+)$ , the set of reachable regions is finitely representable without loosing or introducing behaviour.

### 4.3 A Finite, Complete Prefix for Progressing MTA

In the previous section we have just shown that the reachable regions of the region-stamped automaton of an  $\text{MTA}^+$  are finitely representable. We will use this result to show the existence of a complete, finite prefix of an  $\text{MTA}^+$ 's unfolding.

**Conjecture.** *Let  $\mathcal{A}^+$  be a  $\text{MTA}^+$  with reference clocks  $\mathbf{g}$ . The underlying region automaton  $R_{\mathbf{t}}^{\mathbf{g}}(\mathcal{A}^+)$  can be represented in terms of a finite complete prefix of its unfolding.\**

Unfortunately, we cannot prove this general result yet. But we can provide a proof for a restricted, but practically relevant class of MTA.

The local-region semantics that was defined in Sect. 2.8 introduces a difficulty that was not present in the region semantics of MTA. For an action which has no upper bound in its guard, i.e. which is enabled regardless of how long the system remains in a discrete state, the region-stamped semantics and thereby the local-region semantics provide infinitely many different transitions in the region-automaton. These transitions are distinguished by the amount of global time that has passed since, yielding different successor regions.

In Sect. 4.2 we were able to build the quotient transition system to eliminate these infinitely many regions. The unfolding has no notion of a global state, which yields folding to something like the quotient transition system in the previous section. Furthermore, the local-region semantics introduces more reachable regions that do not correspond to regions of the region automaton. Although we can easily classify those as not reachable if they are not synchronizable (see Theorem 22), they are part of the local-region

automaton. But not-synchronizable regions may lead to synchronizable regions, so we cannot omit the former.

Up to now we haven't found a criterion to fold the not-synchronizable regions such that the state space of the local-region automaton is finitely representable. We therefore have chosen to restrict the class of MTA, where the number of not-synchronizable regions is finite until every (by standard semantics) reachable state has been seen once: We disallow guards that render an action enabled arbitrarily long. Thus in any discrete state, the system either has to do an action after a finite amount of time, or remains in this state forever. Formally, every transition of an MTA  $\mathcal{A} = \{A_p\}_{p \in \mathcal{P}}$  has to contain at least one guard of the form  $x < k$ ,  $x = k$ , or  $x \leq k$ , or more general, the guard has to imply any of these constraints:

$$\forall p \in \mathcal{P} \forall w_p \xrightarrow{\varphi, a, R}_p w'_p \exists x \in C, k \in \mathbb{N} : \varphi \Rightarrow (x \leq k) \quad (5)$$

We call a MTA that satisfies (5) a *progressing MTA*. For this class of MTA, we present our main theorem.

**Theorem 33 (Main Theorem).** *Let  $\mathcal{A}^+$  be a progressing MTA<sup>+</sup> and  $B$ -bounded. The unfolding  $Unf(\mathcal{A}^+) = (E^{\mathfrak{g}}(\mathcal{A}^+), \leq, \#, \lambda)$  has a complete, finite prefix.  $\star$*

*Proof.* Let  $t \in RS(\mathcal{A})$  be a reachable region. There exists a finite ta-sequence  $\rho \in TaSeq_R^{\circ, \mathfrak{g}}(\mathcal{A}^+)$  with  $sim(t, state_R^{\circ, \mathfrak{g}}(\rho))$  by Corollary 10.

Furthermore, there exists a number  $k \in \mathbb{N}$  such that for every reachable  $t \in RS(\mathcal{A})$  exists a finite ta-sequence  $\rho \in TaSeq_R^{\circ, \mathfrak{g}}(\mathcal{A}^+)$  with  $sim(t, state_R^{\circ, \mathfrak{g}}(\rho))$  of length  $|\rho| \leq k$ .  $k$  is bounded by the number of equivalence classes of  $\simeq^{\mathfrak{g}}$  in  $RS^{\mathfrak{g}}(\mathcal{A}^+)$ .

Assume  $t$  has a shortest ta-sequence  $\rho$  that is longer than  $k$ .  $\rho$  has two prefixes  $\rho_1 \sqsubseteq \rho$  and  $\rho_2 \sqsubseteq \rho$ ,  $\rho_1 \neq \rho_2$  that yield equivalent regions:  $state_R^{\circ, \mathfrak{g}}(\rho_1) \simeq^{\mathfrak{g}} state_R^{\circ, \mathfrak{g}}(\rho_2)$ , by the finiteness of  $\simeq^{\mathfrak{g}}$ .

W.l.o.g.  $\rho_1 \sqsubseteq \rho_2$  and let  $\rho = \rho_2 \bar{\rho}_2$ . Then for  $\bar{\rho}_2 \in \uparrow \rho_2$  exists  $\bar{\rho}_1 \in \uparrow \rho_1$  of the same length and  $state_R^{\circ, \mathfrak{g}}(\rho_1 \bar{\rho}_1) \simeq^{\mathfrak{g}} state_R^{\circ, \mathfrak{g}}(\rho_2 \bar{\rho}_2) = state_R^{\circ, \mathfrak{g}}(\rho)$ , by Lemma 32.

We get that  $sim(t, state_R^{\circ, \mathfrak{g}}(\rho_1 \bar{\rho}_1))$  holds, but, by  $\rho_1 \neq \rho_2$ ,  $|\rho_1 \bar{\rho}_1| < |\rho|$ , which contradicts that  $\rho$  was a shortest ta-sequence.

By Proposition 18 and Theorem 22 exists for every reachable  $t \in RS(\mathcal{A})$  a configuration  $K \subseteq E^{\mathfrak{g}}(\mathcal{A}^+)$  of size  $|K| \leq k$  such that for  $state_{R,l}^{\circ, \mathfrak{g}}(K) = s$  with synchronized region  $\vec{s}$  exists the corresponding region  $\bar{s} \in RS^{\mathfrak{g}}(\mathcal{A}^+)$  with  $\vec{s} = \pi_{\mathcal{P}}(\bar{s})$  and  $sim(t, \bar{s})$  holds. Let  $Pref$  be the union of all configurations of  $Unf(\mathcal{A}^+)$  of size  $k + 1$ .

$Pref$  is finite. Trivially,  $Pref$  has finite depth, i.e. the size of prime configurations in  $Pref$  is bounded, by construction. The width of  $Pref$  is finite, i.e.  $Pref$  contains finitely many prime configurations only, because  $\mathcal{A}$  is progressing: let  $K' \subseteq Pref$  be a configuration. In  $s = state_{R,l}^{\circ, \mathfrak{g}}(K')$ , the enabled action  $w_p \xrightarrow{\varphi, a, R}_p w'_p$  allows only a finite number of different time steps  $s \xrightarrow{\delta}_p s''$  after which  $s'' \xrightarrow{a} s'$  can occur. The reason is that  $\varphi$ , containing some formula  $x \leq k$ , can be satisfied only by a finite number of clock regions reachable from  $s$ . The number of actions in  $\mathcal{A}$  is finite, and hence the unfolding can contain only a finite number of events that extend  $K'$ .

$Pref$  is complete. Every reachable region  $t \in RS(\mathcal{A})$  has a configuration in  $Pref$  that yields a state  $s$ , which by Theorem 22 has a stamped region  $\bar{s}$  corresponding to  $s$  with  $sim(t, \bar{s})$ . Because  $Pref$  contains the configurations of size  $k + 1$ , every event that extends a configuration leading to a reachable state  $t$  is part of  $Pref$ . Since these events correspond to the actions enabled in  $t$  (by the definition of  $Unf(\mathcal{A}^+)$ ),  $Pref$  is complete.  $\square$

#### 4.4 Constructing a Complete Finite Prefix

We have just shown the existence of a complete finite prefix of the unfolding of progressing MTA. That existence allows us to think about an algorithm to construct the prefix. We cannot provide such an algorithm yet, but we will present some ideas that helped us to calculate the prefix of some examples. One example is shown in the evaluation in Sect. 5. We use this section to informally present a general idea for an algorithmic solution, and some partial results we consider useful for the definition of an algorithm.

**Prefixes of Unfoldings for Petri nets** McMillan’s idea to construct a complete finite prefix was to identify *cut-off events* in the unfolding after which no more events are added to the prefix. In Petri nets, where unfoldings were applied first, events correspond to transitions. A configuration  $K$  yields the marking that is reached in Petri net if the corresponding transitions of  $K$  are fired in causal order. In a Petri net, when reaching a marking  $m$ , it doesn’t matter which transitions were necessary to reach  $m$ , if one wants to know the possible behaviour of the net after  $m$ . If two configurations  $K$  and  $K'$  yield the same marking  $m$ , the extensions of  $K$  and  $K'$ , respectively, represent the same behaviour of the net (by the firing rule): the extensions are isomorphic.

Therefore, McMillan identifies two configurations as equivalent if they yield the same marking. If during the construction of the prefix (starting at the initial marking), one reaches a configuration  $K$  with a marking that has been seen before in the construction through a configuration  $K'$ , the extensions of  $K$  don’t have to be added to the prefix. McMillan has shown that in Petri nets it is sufficient to consider prime configurations to cut off unnecessary events and still get a complete finite prefix [McM92]. An event of which the prime configuration yields a marking that has been seen before is called a *cut-off event*.

**Requirements for an Algorithmic Solution** For an algorithmic solution to our problem, we suggest a similar approach. We want to identify a criterion for configurations such that we can decide for any MTA  $\mathcal{A}$  – during the construction of the prefix – whether the extensions of a configuration are relevant for a complete finite prefix of its unfolding  $Unf(\mathcal{A})$ , or not. This criterion needs to associate two configurations with each other if they yield isomorphic (or bisimilar) extensions. This nature of the criterion implies that it is an equivalence relation. If two such equivalent configurations are found and one of them with all of its extending events is already part of the prefix, then the extending events of the other configuration needn’t to be part of the prefix.

The construction algorithm then must omit the extensions of a configuration only if the state, that is reached by the configuration, hasn't been seen before in the prefix; thus the prefix becomes complete. The criterion itself should be an equivalence relation of finite index to ensure that after finitely many configurations (of finite size), all further configurations yield no new future behaviour and hence their extensions can be omitted. The proof of Theorem 33 shows that such a method yields a complete, finite prefix of  $Unf(\mathcal{A})$ .

What we have to do is to find an equivalence relation on configurations with the desired property that equivalent configurations have isomorphic extensions; furthermore this equivalence relation has to have a finite index.

**A Solution for MTA** Currently, we suggest to use a criterion based on the state that is reached in a configuration of  $Unf(\mathcal{A})$ : define an equivalence relation on decomposed regions of  $\mathcal{A}^+$  such that any two configurations that yield equivalent states have isomorphic (or bisimilar) extensions. We think that there exists such a criterion for decomposed regions. Let's consider the criteria for the region automaton and the stamped-region automaton first.

In the region automaton  $R(\mathcal{A})$  of a MTA  $\mathcal{A}$ , after reaching a region  $s$ , the future behaviour in  $R(\mathcal{A})$  depends on  $s$  only: for any two ta-sequences  $\hat{\rho}_1, \hat{\rho}_2 \in TaSeq_R^\circ(\mathcal{A})$  with  $state_R^\circ(\hat{\rho}_1) = state_R^\circ(\hat{\rho}_2)$  the set of all possible extensions  $\uparrow\hat{\rho}_1$  and  $\uparrow\hat{\rho}_2$  are isomorphic, which follows directly from the definition of a step in  $R(\mathcal{A})$  (see Sect. 2.7). The equivalence relation on the states is given here by equality.

A similar result holds for the region-stamped automaton  $R^g(\mathcal{A}^+)$  because of the bisimilarity (see Lemma 31). We may use the equivalence relation  $\simeq^g$  on stamped regions which we found in Sect. 4.3: for any two ta-sequences  $\hat{\rho}_1, \hat{\rho}_2 \in TaSeq_R^{\circ,g}(\mathcal{A}^+)$  with  $state_R^{\circ,g}(\hat{\rho}_1) \simeq^g state_R^{\circ,g}(\hat{\rho}_2)$  the set of all possible extensions  $\uparrow\hat{\rho}_1$  and  $\uparrow\hat{\rho}_2$  are bisimilar. In the quotient transition system  $[R^g(\mathcal{A}^+)]_{\simeq^g}$  of  $R^g(\mathcal{A}^+)$  under the stamped region equivalence  $\simeq^g$ , the extensions of the corresponding ta-sequences are isomorphic, c.f. Lemma 32.

A likewise expressive criterion for isomorphic extensions of the local-region automaton, and hence the unfolding of  $\mathcal{A}$ , is more tricky; and we haven't found such a criterion, that has the desired property in any case, yet. But we have a partial result, which we would like to present here briefly.

Observe that within a single component  $A_p^+$  of an MTA<sup>+</sup>  $\mathcal{A}^+$ , two regions with the same discrete state, equivalent clock regions, and buffers with the same number of messages (in which all region stamps are less than or equal to the value of the reference clock), enable the same set of region-stamped actions. Observe further that in any synchronized, reachable, decomposed region, the region stamps in the buffers are less than or equal to the value of the reference clocks.

Thus in two decomposed, synchronized regions  $\vec{s} = ((w_1, \vec{\alpha}_1), \dots, (w_P, \vec{\alpha}_P), \beta)$ ,  $\vec{s}' = ((w'_1, \vec{\alpha}'_1), \dots, (w'_P, \vec{\alpha}'_P), \beta) \in RS_{R,l}^{\circ,g}(\mathcal{A}^+)$  with

$$\begin{aligned} \forall p \in \mathcal{P} : w_p = w'_p \wedge \vec{\alpha}_p \sim_{r,\Delta} \vec{\alpha}'_p \\ \wedge \forall q \in \mathcal{P} : p \neq q \Rightarrow \forall m \in M(p, q) : |\beta(p, q, m)| = |\beta'(p, q, m)|, \end{aligned} \quad (6)$$

the same sets of region-stamped actions are enabled. From Theorem 22 and Lemma 32 follows that two configurations  $K, K' \subseteq E^{\circ, \mathbf{g}}(\mathcal{A}^+)$  that yield  $state_{R,l}^{\circ, \mathbf{g}}(K) = \vec{s}$  and  $state_{R,l}^{\circ, \mathbf{g}}(K') = \vec{s}'$  satisfying criterion (6) have bisimilar extensions. Unfortunately, only few configurations yield synchronized regions, but the majority has *synchronizable* regions.

In case the configurations  $K, K'$  yield  $state_{R,l}^{\circ, \mathbf{g}}(K) = s$  and  $state_{R,l}^{\circ, \mathbf{g}}(K') = s'$  being synchronizable such that the catch-up regions  $\vec{s}$  and  $\vec{s}'$  are equivalent by (6), then the extensions *after*  $\vec{s}$  and  $\vec{s}'$  are isomorphic. This can be translated to configurations and extending events in the following way:

For a configuration  $K$  let  $\uparrow K$  be the least set of events such that

- if  $e$  is an extension of  $K$ , then  $e \in \uparrow K$ , and
- if  $e$  is an extension of  $K \uplus \{e_1, \dots, e_n\}$  with  $\{e_1, \dots, e_n\} \subseteq \uparrow K$  then  $e \in \uparrow K$ ,

holds; i.e.  $\uparrow K$  is the set of events that lies completely after  $K$ , and thereby corresponds to the set of extensions  $\uparrow \rho$  of a ta-sequence  $\rho$  which leads to the same region as  $K$ . Let  $K$  be such that  $state_{R,l}^{\circ, \mathbf{g}}(K) = ((w_1, \alpha_1), \dots, (w_P, \alpha_P), \beta) = s$  is synchronizable. By

$$\uparrow K^{\max} = \left\{ e \in \uparrow K \mid \lambda(e) = (a, \tau), \tau \geq \max_{p \in P} \alpha_p(g_p) \right\}$$

denote the set of events extending  $K$  that occur after the latest event in  $K$ . These events correspond to region-stamped actions  $(a, \tau)$  that are enabled in the synchronized region  $\vec{s}$  of  $s$ .

We achieve the result that for any two configurations  $K, K' \subseteq E^{\circ, \mathbf{g}}(\mathcal{A}^+)$  that yield the synchronizable regions  $state_{R,l}^{\circ, \mathbf{g}}(K) = s$  and  $state_{R,l}^{\circ, \mathbf{g}}(K') = s'$  where the corresponding catch-up regions  $\vec{s}$  and  $\vec{s}'$  satisfy (6), the sets  $\uparrow K^{\max}$  and  $\uparrow K'^{\max}$  are bisimilar, i.e. yield equal time-abstract sequences of actions. We think that in analogy to the quotient transition system under  $\simeq^{\mathbf{g}}$  we defined in Sect. 4.2, there exists some quotient in  $Unf(\mathcal{A})$  due to  $\sim_{r, \Delta}$  that yields isomorphic extensions.

The criterion above spares two important cases:

1. In case of configurations  $K, K'$  yielding synchronizable states, when are the full sets of extensions  $\uparrow K$  and  $\uparrow K'$  bisimilar or isomorphic?
2. When are we allowed to cut off a not synchronizable configuration?

Finally observe an important technicality: since synchronizable regions are usually not reached by prime configurations, our criterion does not define cut-off events, but *cut-off configurations*. We assume that this has a serious impact on how the algorithm to construct the finite, complete prefix needs to be designed.

The answer to these questions as well as the search for an algorithmic solution is left as future work.

## 5 Evaluation of the Reduction Method

Before we draw a conclusion of our suggested technique, we would like to evaluate its usefulness in solving our problem to reduce the state space explosion due to concurrently enabled actions. In this thesis we provided the theoretical foundation for our technique. This foundation, namely the unfolding of a MTA and our cut-off criterion, allows us to manually calculate a complete, finite prefix of an unfolding of a small example network.

To evaluate our method we proceeded as follows. We have chosen a progressing MTA  $\mathcal{A}_3$ , depicted in Fig. 13, with concurrently enabled actions that is small enough to be analyzed manually; the MTA is explained in Sect. 5.1. We then determined the region equivalence relation  $\sim_r$  for the clock variables of  $\mathcal{A}_3$  and calculated the region graph which represents all clock regions and how they are related to each other. This graph is necessary to compute the region automaton  $R(\mathcal{A}_3)$  that finitely represents the state space of  $\mathcal{A}_3$ , which we did afterwards. The region automaton  $R(\mathcal{A}_3)$  is what would be calculated if  $\mathcal{A}_3$  was model checked and it suffers the state space explosion because of concurrently enabled actions. The results of the calculation are presented in Sect. 5.2.

The next step in the evaluation was to compute a complete, finite prefix of the unfolding  $Unf(\mathcal{A}_3)$  of  $\mathcal{A}_3$ . For this computation, we determined the alternative region equivalence relation  $\sim_{r,\Delta}$  for each component of  $R(\mathcal{A}_3)$ , which is necessary to apply the cut-off criterion (6) that we suggested in Sect. 4.4. We will explain how we constructed the prefix in more detail in Sect. 5.3, where we will also show how the prefix represents the behaviour of  $\mathcal{A}_3$ . This prefix is what an algorithmic solution that bases on our approach would calculate; it is meant to alleviate the state space explosion.

We then analyzed the determined prefix of  $Unf(\mathcal{A}_3)$  and compared it to  $R(\mathcal{A}_3)$ . During the analysis, we found that the prefix contains some further redundant information which can be removed in a systematic way which results in a smaller prefix of  $Unf(\mathcal{A}_3)$ . We then compared the smaller prefix of  $Unf(\mathcal{A}_3)$  with  $R(\mathcal{A}_3)$  to estimate amount of reduction of the state space that is achievable by our method. The conclusions of the analysis are given in Sect. 5.4.

Regarding notation: Because it is the only prefix that we will consider in this section, we allow ourselves to use the single word ‘prefix’ when speaking of the complete, finite prefix of  $Unf(\mathcal{A}_3)$ .

### 5.1 A progressing MTA

The system we will consider for the remainder of this section is depicted in Fig. 13. The MTA  $\mathcal{A}_3$  consists of two components  $A_l$  and  $A_r$  that communicate by a simple ping-pong protocol. We describe the intended behaviour to provide an intuitive understanding of how its state space, and thereby its region automaton and its unfolding, are structured. The system will allow more behaviour than the intended one, which is on purpose to make the state space “more interesting”.

Each component runs rounds, where a round needs at most one time unit. In either component, the first action ( $a$  in  $A_l$  and  $b$  in  $A_r$ ) is dedicated to initialize the round by

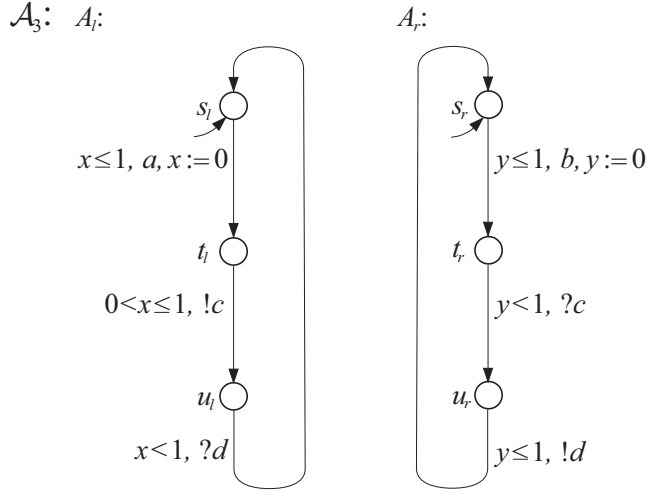


Figure 13: A progressing MTA  $\mathcal{A}_3$ .

resetting the clock variable ( $x$  in  $A_l$  and  $y$  in  $A_r$ , respectively); the initialization may happen concurrently. Then  $A_l$  sends a message to  $A_r$  (action  $!c$ )<sup>10</sup>, “ping”, which is received (action  $?c$ ) and replied (action  $!d$ ) by  $A_r$ , “pong”; then  $A_r$  finishes its round. The reply has to be sent in time to allow  $A_l$  to receive the reply (action  $?d$ ) and finish its round as well.

A “good” ta-sequence of  $\mathcal{A}_3$  is

$$(b, [0; 0]) (a, (0; 1)) (!c, (0; 1)) (?c, (0; 1)) (!d, [1; 1]) (?d, [1; 1]) (b, [1; 1]) (a, [1; 1]) \quad (7)$$

while a “bad” ta-sequence of  $\mathcal{A}_3$  is

$$(b, [0; 0]) (a, (0; 1)) (!c, [1; 1]) \quad (8)$$

which leads to a deadlock where  $A_r$  cannot receive the “ping” because the guard  $y < 1$  of action  $?c$  is no longer satisfied.

## 5.2 The State Space

In this section, we briefly present and explain the region automaton  $R(\mathcal{A}_3)$  of  $\mathcal{A}_3$ . The definition of the region automaton of a MTA requires to determine the region equivalence relation  $\sim_r$  of the system first. This yields the *region graph*, describing how all possible clock regions of  $\mathcal{A}_3$  are related to each other by the progress of time and the reset of clock variables. With the abstract representation of time at hand, we could construct the region automaton of  $\mathcal{A}_3$  with a depth-first search algorithm that follows the possible combined steps according to Sect. 2.8. Both, the region graph and the region automaton have been calculated automatically using a small self-written tool.

The region automaton is easier to understand if one knows the region graph of the system. We therefore show the region graph in Fig. 14.

<sup>10</sup>We use the short-cut notation  $!c$  for  $!r(c)$  and  $?c$  for  $r?l(c)$ , similarly for  $!d$  and  $?d$

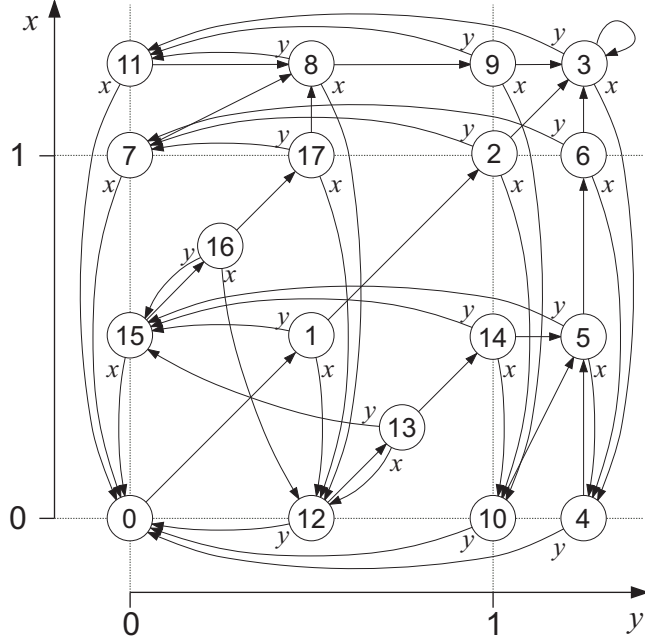


Figure 14: The region graph of  $\mathcal{A}_3$  from Fig. 13

Each node in Fig. 14 represents a clock region of  $\mathcal{A}_3$ , e.g. 0 represents the region  $x = 0 \wedge y = 0$ , i.e.  $\{\sigma \in \Sigma(\{x, y\}) \mid \sigma \models x = 0 \wedge y = 0\}$ . The arrows show their relations:

1. Unlabelled arrows point from a region  $\alpha$  to its direct successor region  $\text{succ}_r(\alpha)$ , e.g. 0 to 1, which represents  $0 < x < 1 \wedge 0 < y < 1 \wedge \text{fract}(x) = \text{fract}(y)$ .
2. Arrows labelled with a clock variable  $x$  point from a region  $\alpha$  to the region  $\alpha[\{x\} \mapsto 0]$ , e.g. 1 to 12 representing  $0 < y < 1 \wedge x = 0$

The nodes have been arranged such that the clock constraints can be deduced from their position in the diagram, e.g. node 12. The nodes beyond clock value 1 represent clock regions where the variable in the respective dimension exceeds  $\mathcal{A}_3$ 's maximal constant against which that variables is compared, e.g. node 4 where  $y > k_y = 1$ . We will refer the clock regions by the corresponding node labels, e.g.  $\alpha^1$  refers to node 1.

The numbering scheme of the region graph in Fig. 14 has been used to annotate the regions in the region automaton  $R(\mathcal{A})$  that is depicted in Fig. 16.<sup>11</sup> Each node in the graph of Fig. 16 represents a reachable region of  $\mathcal{A}_3$ . The nodes are labelled with the region information, i.e. with the vector of the discrete states, followed by the number of the clock region according to Fig. 14 and finally the contents of the buffers. For example, the initial region  $((s_l, s_r), \alpha^0, \beta^0)$  is labelled  $slsr0$ . Each edge represents a combined step and is labelled with the step's action. The “good” run (7) has been highlighted in the figure.

<sup>11</sup>The graphical representation was generated automatically by the Graphviz-toolsuite, <http://www.graphviz.com>.

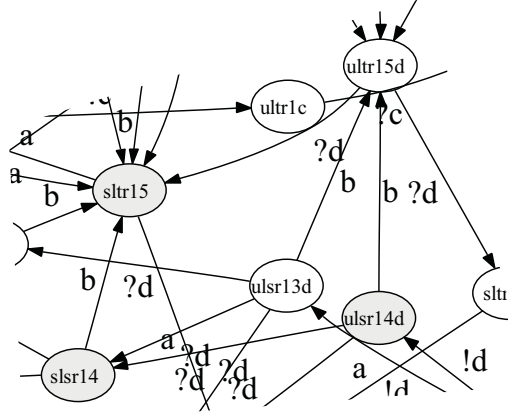


Figure 15: A diamond in  $R(\mathcal{A}_3)$  because of concurrently enabled events  $b$  and  $?d$ .

The region automaton  $R(\mathcal{A}_3)$  contains 57 regions. Furthermore, we can identify some diamond structures due to concurrent events. For instance  $?d$  and  $b$  are concurrently enabled in region  $ulsr14d$ ; we could execute

$$ulsr14d (?d, [1; 1]) slsr14 (b, [1; 1]) sltr15$$

as well as

$$ulsr14d (b, [1; 1]) ultr15d (?d, [1; 1]) sltr15.$$

Figure 15 shows the diamond in  $R(\mathcal{A}_3)$ . However, a very large contribution to the size of the state space roots in the presence of time in the states. One can already see from the region graph in Fig. 14 that there are exponentially many (in the number of clocks) different clock regions, which occur in  $R(\mathcal{A}_3)$ .

### 5.3 A Finite Complete Prefix of its Unfolding

The next step in our evaluation was to calculate a complete, finite prefix of the unfolding of  $\mathcal{A}_3$ . For the lack of a tool, we had to calculate the prefix of  $\mathcal{A}_3$ 's unfolding by hand. We did so by first constructing an initial part of the unfolding by following the definition of  $Unf(\mathcal{A}_3)$ , see Sect. 2.9.2. In a next step, we identified cut-off configurations according to the criterion (6) for synchronized configurations which we explained in Sect. 4.4.

**Method of Construction** The prefix was generated from the initial part of the unfolding as follows: By a breadth-first search on the unfolding, we marked events to be part of the prefix. During the exploration of  $Unf(\mathcal{A}_3)$ , upon marking another event to be part of the prefix, we identified configurations that included the newly marked event and that yield a synchronized region. We then searched for configurations of marked events that yield an equivalent synchronized region. The equivalence was decided by the criterion (6) of Sect. 4.4 and using the region graph of Fig. 17 (see below). If an equivalent pair of configurations was found, the new configuration  $K$  was considered as cut-off



configuration: behaviour after  $K$  can be omitted because it is bisimilar to the behaviour after  $K'$  (see Sect. 4.4). This means that any event that extends  $K$  was not explored.

Observe that  $K$  is in general the union of several maximal prime configurations, where some of these are part of another configuration  $K' \neq K$ . If  $K'$  is not identified as a cut-off configuration as well then the events that extend  $K'$  but not  $K$  have to be explored. Hence this algorithm terminates when beyond a certain point for each event  $e$  that is added to the prefix, all possible configurations that include  $[e]$  as a causally maximal prime configuration are considered as a cut-off configuration. The proof of Theorem 33 shows that this eventually happens.

After we obtained a finite, complete prefix, we re-searched the remaining configurations for more cut-off configurations to obtain a minimal prefix. Although we have no proof, we think that the prefix of  $Unf(\mathcal{A}_3)$  that is depicted in Fig. 18 is minimal wrt the cut-off criterion (6).

**Deciding Equivalence** The cut-off criterion involves region equivalence  $\sim_{r,\Delta}$  (see (6) in Sect. 4.4). When applying the cut-off criterion we had to decide whether two clock regions in the MTA<sup>+</sup>  $\mathcal{A}_3^+ = \{A_l^+, A_r^+\}$  with reference clocks  $g_l$  and  $g_r$  are equivalent. Proposition 29 in Sect. 4.1 shows that we can do this by the help of the region graphs of  $A_l^+$  and  $A_r^+$  that includes the reference clocks.

The clock variables are  $C_l^+ = \{x, g_l\}$  and  $C_r^+ = \{y, g_r\}$ . The region graph of  $A_l^+$  is depicted in Fig. 17, the graph of  $A_r^+$  is isomorphic to Fig. 17; the graph for  $A_l^+$  has been constructed and laid out like the region graph of  $\mathcal{A}_3$  in Fig. 14. Because the reference clock  $g_l$  is never reset, the region graph has a different structure compared to the one in Fig. 14. It is infinite and unbounded in the dimension of  $g_l$ ; for the construction of the prefix it is sufficient to consider the depicted part of the region graph ( $\mathcal{A}_3$  being progressing prevents time to advance arbitrarily far before a new round begins).<sup>12</sup>

We will use the node numbers of Fig. 17 to refer to clock regions of  $A_l^+$  as well as  $A_r^+$ . Observe that the region graph exhibits a regularity: The graph from node 23 onwards (the reachable nodes by following the arcs forward) is structurally isomorphic to the graph from node 0 onwards. Remember further that two clock regions  $\alpha, \alpha' \in R(C_l^+)$  are equivalent by  $\sim_{r,\Delta}$  if their projections  $\Pi_{C_l}(\alpha)$  and  $\Pi_{C_l}(\alpha')$  are equal. The equivalence can easily be seen in the figure by projection onto the  $x$ -axis. For instance the clock regions depicted by nodes 2 and 30 are equivalent. It is similarly easy to see from the diagram in Fig. 17 whether two clock regions are synchronized, e.g. nodes 30 and 29 both satisfy  $1 < g_l < 2$ .

We refer to clock regions of  $A_l^+$  by symbols  $\alpha_l^i$  where  $i$  is the number of the node in Fig. 17 that corresponds to the clock region; similarly  $\alpha_r^i$  denotes a clock region of  $A_r^+$ .

**A graphical Representation** A complete (and obviously finite) prefix of  $Unf(\mathcal{A}_3)$  is shown in Fig. 18. To support the understanding, we use the notation of an *occurrence net*, which enriches the notion of the event structure with *conditions* that represent parts

<sup>12</sup>The numbering scheme of the nodes in Fig. 17 originates from the application of a self-written tool and has no further interpretation.



3. The occurrence of action  $a$  in the initial states yields event  $e_2$ , and results in the region  $s^2 = ((t_l, \alpha_l^{26}), (s_r, \alpha_r^0), \beta^0)$ . The part of the state that has changed by this occurrence is the local region of  $A_l^+$ , thus a condition labelled with  $(t_l, \alpha_l^{26})$  is added as the *post-condition* of  $e_2$  with a connecting arc.
  4.  $!c$  can occur in  $s^2$ .  $(t_l, \alpha_l^{26})$  is the pre-condition of the corresponding event  $e_{57}$ . The occurrence of  $!c$  produces a message in buffer  $(l, r, c)$  and yields the region  $s^{57} = ((u_l, \alpha_l^{29}), (s_r, \alpha_r^0), \beta^{57})$  where  $\beta^{57}(l, r, c) = [(1; 2)]$ . Hence, conditions that are labelled with  $(u_l, \alpha_l^{29})$  and  $(1; 2)$  are added as post-conditions of  $e_{57}$ .
- The post-conditions of a configuration represent the region that is reached by this configuration.

**Example of Configurations and Cut-Off Configurations** To increase the understanding of our construction, we want to present a small example of a cut-off configuration and how configurations correspond to runs in the region automaton.

Let's locate the "good" run (7) in the prefix of Fig. 18. The nodes in the figure have been shaded: white nodes represent events and conditions that belong to  $A_l^+$ , light grey nodes represent events and conditions that belong to  $A_r^+$ , dark grey nodes represent message-conditions that belong to neither component. The configuration that corresponds to the first four actions

$$(b, [0; 0]) (a, (0; 1)) (!c, (0; 1)) (?c, (0; 1))$$

of the run is

$$K_1 := \{e_8, e_2, e_{55}, e_{58}\}.$$

The post-conditions of  $K_1$  are labelled with  $(u_l, \alpha_l^{27})$  and  $(u_r, \alpha_r^1)$ . This represents the synchronized region  $s^1 = ((u_l, \alpha_l^{27}), (u_r, \alpha_r^1), \beta^0)$ . The region  $\bar{s}^1 = ((u_l, u_r), \bar{\alpha}^{13}, \bar{\beta}^0) \in RS(\mathcal{A}_3)$  – with  $\bar{\alpha}^{13}$  the clock region that is represented by node 13 in Fig. 14 – corresponds to  $s^1$ ; see node  $ulur13$  in Fig. 16.

During the calculation of the prefix, we identified  $K_1$  as a cut-off configuration. Because  $\alpha_l^{27} \sim_{r, \Delta} \alpha_l^1$  holds,  $K_1$  corresponds to

$$K_2 := \{e_8, e_1, e_{15}, e_{21}\}$$

which we had seen "before" while we analyzed the prefix.  $K_2$  yields the decomposed region  $s^2 = ((u_l, \alpha_l^1), (u_r, \alpha_r^1), \beta^0)$  for which as well  $s^2 = \pi_{\mathcal{P}}(\bar{s}^1)$  holds, because the relative constraints between clocks of  $A_l$  and  $A_r$  are projected out.

The remaining part of the ta-sequence (7) is

$$(!d, [1; 1]) (?d, [1; 1]) (b, [1; 1]) (a, [1; 1])$$

with corresponding events

$$K_3 := \{e_{34}, e_{41}, e_{50b}, e_{49b}\}$$

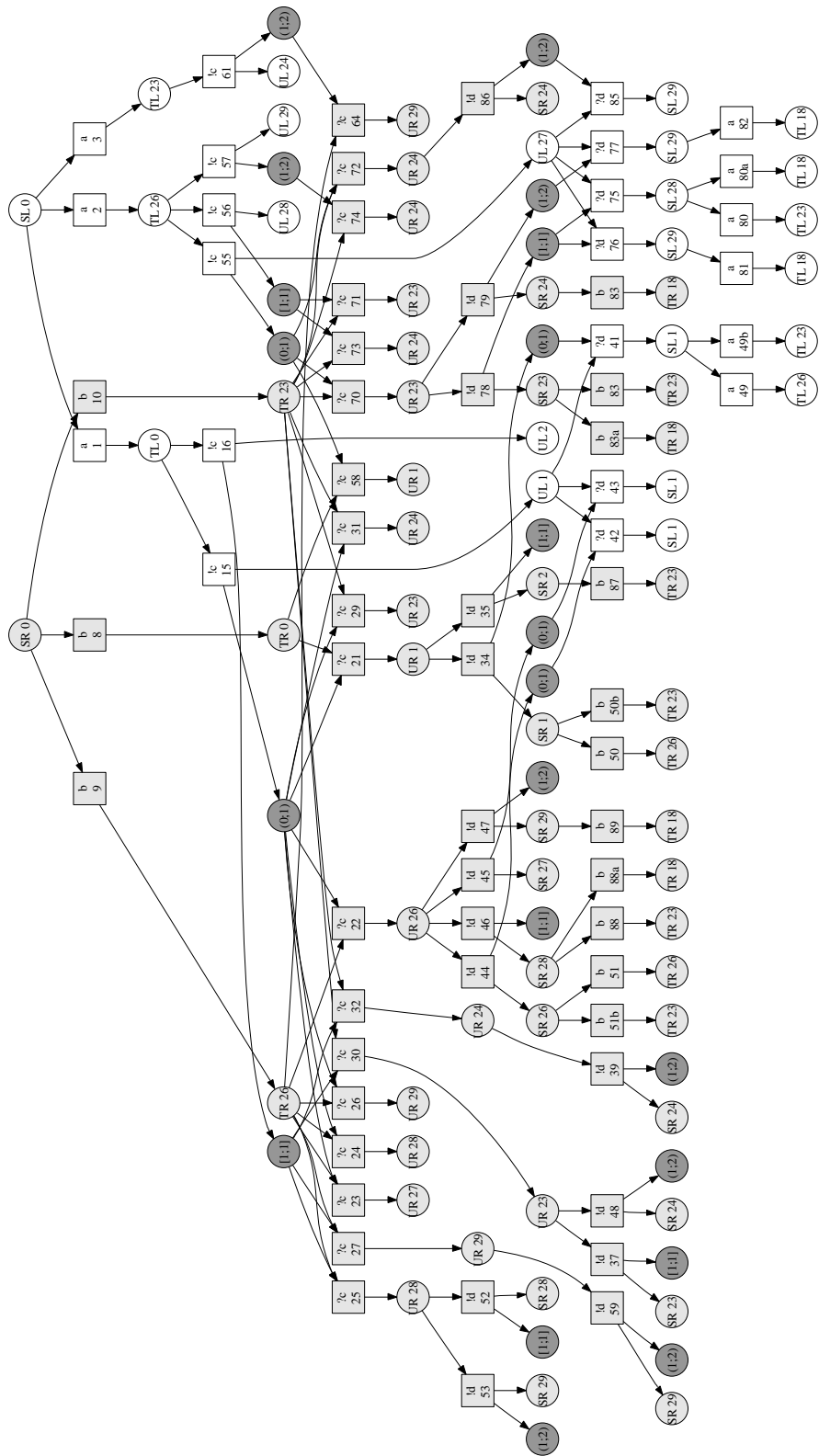


Figure 18: A complete finite prefix of  $Unf(\mathcal{A}_3)$ .

that extend  $K_2$ . We can see that the configuration  $K_2 \uplus K_3$  yields the decomposed region  $s^3 = ((t_l, \alpha_l^{23}), (t_r, \alpha_r^{23}), \beta^0)$ . The corresponding region to  $s^3$  in the region graph is  $\bar{s}^3 = ((t_l, t_r), \bar{\alpha}^0, \bar{\beta}^0)$ , which is also the resulting state of the run in  $R(\mathcal{A}_3)$ , see node  $tltr0$  in Fig. 16.

Observe that the diamond that is caused by  $(?d, [1; 1])$  and  $(b, [1; 1])$ , and that requires four transitions and four region (see Sect. 5.2) in the region automaton, is represented here succinctly by events  $e_{50b}$  and  $e_{41}$  only. The configuration  $K_2 \uplus K_3$  yields a synchronized region and is a cut-off configuration that corresponds, for instance, to  $\{e_1, e_8\}$  because  $\alpha_l^{23} \sim_{r, \Delta} \alpha_l^0$  holds.

Similarly, the “bad” run  $(7) - (b, [0; 0]) (a, (0; 1)) (!c, [1; 1])$  is represented in  $Unf(\mathcal{A}_3)$  by the configuration

$$\{e_8, e_2, e_{56}\}$$

which is no cut-off configuration and has no extension in  $Unf(\mathcal{A}_3)$ .

## 5.4 Analysis of the Prefix

In the final part of this section, we present the results of our analysis, whether the prefix is a better representation of the state space of a MTA than its region automaton. Both,  $R(\mathcal{A}_3)$  and  $Unf(\mathcal{A}_3)$ , use regions to finitely represent the timing information. The amount of reduction may only be due to concurrently enabled events, which are ordered totally in  $R(\mathcal{A}_3)$  and ordered partially in  $Unf(\mathcal{A}_3)$ .

**Space Complexity** At the moment, the numbers of regions and events (and conditions) cannot be compared directly to obtain a quantitative statement because actual space requirements depend on an implementation. But they are sufficient for some qualitative reasoning in terms of the  $\mathcal{O}$ -notation.

The prefix shown in Fig. 18 consists of 69 events and 92 conditions. The minimal structure that is necessary to represent the prefix contains the 69 events. The conditions follow from the causality relation between events (and could be computed whenever necessary). The region automaton contains 57 reachable regions, as state above.

We have no further case studies yet, but obviously, the prefix and the region automaton have the same space complexity: they are exponential in the size of the original system. Now remember that the representation of time in the region abstraction already introduces exponentiality to the region automaton. The unfolding does not address this reason of exponentiality and hence, it does not reduce exponentiality due to clock regions.

The prefix itself explodes because for any action  $a$  that is enabled in a region  $s$ , all combined steps  $s \xrightarrow{\delta, a}_p s'$  that yield different successor regions are conflicting. Hence, in the unfolding each action causes several conflicting events. Since by definition, conflicting events have no common successors, the unfolding grows exponentially in the number of actions. The amount of conflicting events in the given automaton is far larger than the amount of concurrent events. It seems that the explosion due to conflicting events is larger than the reduction due to concurrent events. Of course, conflicting transitions in

$R(\mathcal{A})$  contribute to the explosion as well, but to a lesser extend, since two conflicting steps (due to the same action) may lead to the same region later on.

This leads to the conjecture that in systems, where an action can cause only few conflicting events, while it is concurrently enabled to a large set of actions in other components, the benefit of the reduction is larger than the explosion due to conflicting events. This conjecture needs to be validated experimentally.

**Improving Space Complexity** We now want to point out a possible improvement in the method that may yield a smaller prefix by reducing the explosion due to conflicting events.

We had to introduce reference clocks into the system for the construction of the unfolding in order to avoid that a message may be received before it is sent, because we de-synchronized the progress of time. We have shown through the alternative region equivalence  $\sim_{r,\Delta}$  that this additional clock does no harm to the behaviour of an MTA (see Proposition 29 and Theorem 22).

However, the reference clock creates an overhead in the unfolding, which is not necessary to equivalently express the set of reachable regions. Observe that in the region graph of Fig. 17, the nodes 27, 28 and 29 are subsequent clock regions, but have the same clock constraint on  $x$  ( $0 < x < 1$ ). They differ in the constraints on the reference clock only: the clock regions are pairwise region equivalent, but *not* synchronized.

Therefore the guard of an action is satisfied in  $\alpha_l^{27}$  iff it is satisfied in  $\alpha_l^{28}$  iff it is satisfied in  $\alpha_l^{29}$ . Additionally, for receive actions, if the message can be received in  $\alpha_l^{27}$ , it can be received in  $\alpha_l^{28}$  and  $\alpha_l^{29}$ . Because the clock regions are subsequent, they yield several conflicting events (up to 3) for the same action, where the region automaton provides just a single transition. For instance, consider the extending events  $e_{55}, e_{56}, e_{57}$  of  $[e_2]$  in Fig. 18 and the arc  $tl sr12 \xrightarrow{!c} ul sr14c$  in Fig. 16. The extending events have equivalent post-conditions  $((u_l, \alpha_l^{27}), (u_l, \alpha_l^{28})$  and  $(u_l, \alpha_l^{29}))$  that all, together with  $(s_r, \alpha_r^0)$  and some message condition, correspond to the same region  $((u_l, s_r), \bar{\alpha}^{14}, \bar{\beta})$  in  $R(\mathcal{A}_3)$ .

We observed a similar property in region-stamped automata. The equivalence relation  $\simeq^{\mathfrak{g}}$  on stamped regions, which we found in Sect. 4.2 is a non-trivial bisimulation relation from a region-stamped automaton to itself, where ‘non-trivial’ means that it is distinct from identity. Remember that  $\simeq^{\mathfrak{g}}$  related two stamped regions as equivalent if they differed in the value of the reference clock only; this is the same phenomenon that we face in our unfolding.

In Lemma 32 we have shown that the quotient transition system of a region-stamped automaton removes exactly this redundancy by folding equivalent states into their equivalence class. This folding is due to  $\sim_{r,\Delta}$  only (see definition of  $\simeq^{\mathfrak{g}}$ ). For this reason, we applied the same folding operation to the prefix in Fig. 18. We relabelled all conditions that represent local regions  $(w, \alpha)$  with  $(w, \alpha')$  where  $\alpha' \sim_{r,\Delta} \alpha$  and  $\alpha'(g)$  is minimal. The remaining clock regions in the thereby re-labelled prefix are  $\alpha_p^i$ , with  $p \in \{l, r\}$  and  $i \in \{0, \dots, 3\}$ .

One can see that conflicting events  $f_1, \dots, f_m$  with the same pre-conditions and equally

labelled post-conditions have bisimilar extensions: The respective extensions  $E_1, \dots, E_m$  of  $\lfloor f_1 \rfloor, \dots, \lfloor f_m \rfloor$  represent the same runs in  $R(\mathcal{A}_3)$ . This observation is supported by our cut-off criterion (6). We then folded equally labelled post-conditions of the same conflicting event into an “equivalence class” of conditions; and the subsequent events and conditions into respective equivalence classes of events and conditions, this operations is sound because of the bisimilarity of the branches. We show the result of this re-labelling and folding operation in Fig. 19.

In the folded prefix, the events  $e_{55}$ ,  $e_{56}$ , and  $e_{57}$  are represented by  $e_{55}$  only. The configuration  $K_2 \uplus K_3 = \{e_8, e_1, e_{15}, e_{21}, e_{34}, e_{41}, e_{50b}, e_{49b}\}$  that represents the “good” run (7) is now folded to

$$\{e_8, e_1, e_{21}, e_{15}, e_{34}, e_{41}, e_{50}, e_{49}\}.$$

The folded prefix in Fig. 19 contains a mere 47 events and 64 conditions while it represents the same set of reachable regions as the region automaton  $R(\mathcal{A}_3)$  that contains 57 reachable regions as mentioned above. The folded prefix is significantly smaller (by factor 1.5) than the original prefix and – depending on the implementation – might be smaller than the region automaton.

This folding operation on the given example suggests that the construction of a prefix that equivalently represents the runs and reachable states of the region automaton can be improved. We think that there exists a labelled event structure that enjoys the equivalence to the region automaton, and which *is smaller* than the region automaton.

Though, the folded prefix is still exponential in the size of the original system. One reason remains the exponential amount of clock regions that are also part of the state space, but it seems as if the state space explosion due to concurrent events can be alleviated without introducing a new source of exponentiality.

As a final remark, it is worth to consider the space complexity of the unfolding in more detail. In the region automaton, a region contains a clock region over *all* clocks of all components. This leads to more than  $\mathcal{O}(|C|! \cdot 2^{|C|})$  many different clock regions that may occur in the region automaton<sup>14</sup> In the unfolding, the events may yield only  $\mathcal{O}(|C_p^+|! \cdot 2^{|C_p^+|}) = \mathcal{O}((|C_p| + 1)! \cdot 2^{|C_p|+1})$  many different clock regions per component  $p \in \mathcal{P}$ . This leads to  $\mathcal{O}((|C_1| + 1)! \cdot 2^{|C_1|+1}) + \dots + \mathcal{O}((|C_P| + 1)! \cdot 2^{|C_P|+1})$  different clock regions that may occur in the unfolding, which is significantly less than  $\mathcal{O}(|C|! \cdot 2^{|C|})$  since  $|C| = |C_1| + \dots + |C_P|$ .

This especially reduces the amount of conflicting events due to successor regions in the unfolding, not by a factor, but by magnitude! We could not observe this effect in the example, because the overhead of the reference clock countered the reduction due to less clocks per clock region. Further case studies are necessary to validate this effect.

Finally, with fewer clocks per clock region, the amount of memory that is necessary to store a clock region in an implementation is reduced: the clock constraints between

---

<sup>14</sup>The actual upper bound of clock regions also depends exponentially on the length of all clock constrains that occurs in the automaton, which increases the amount of clock regions even more. For a detailed analysis see [AD94].

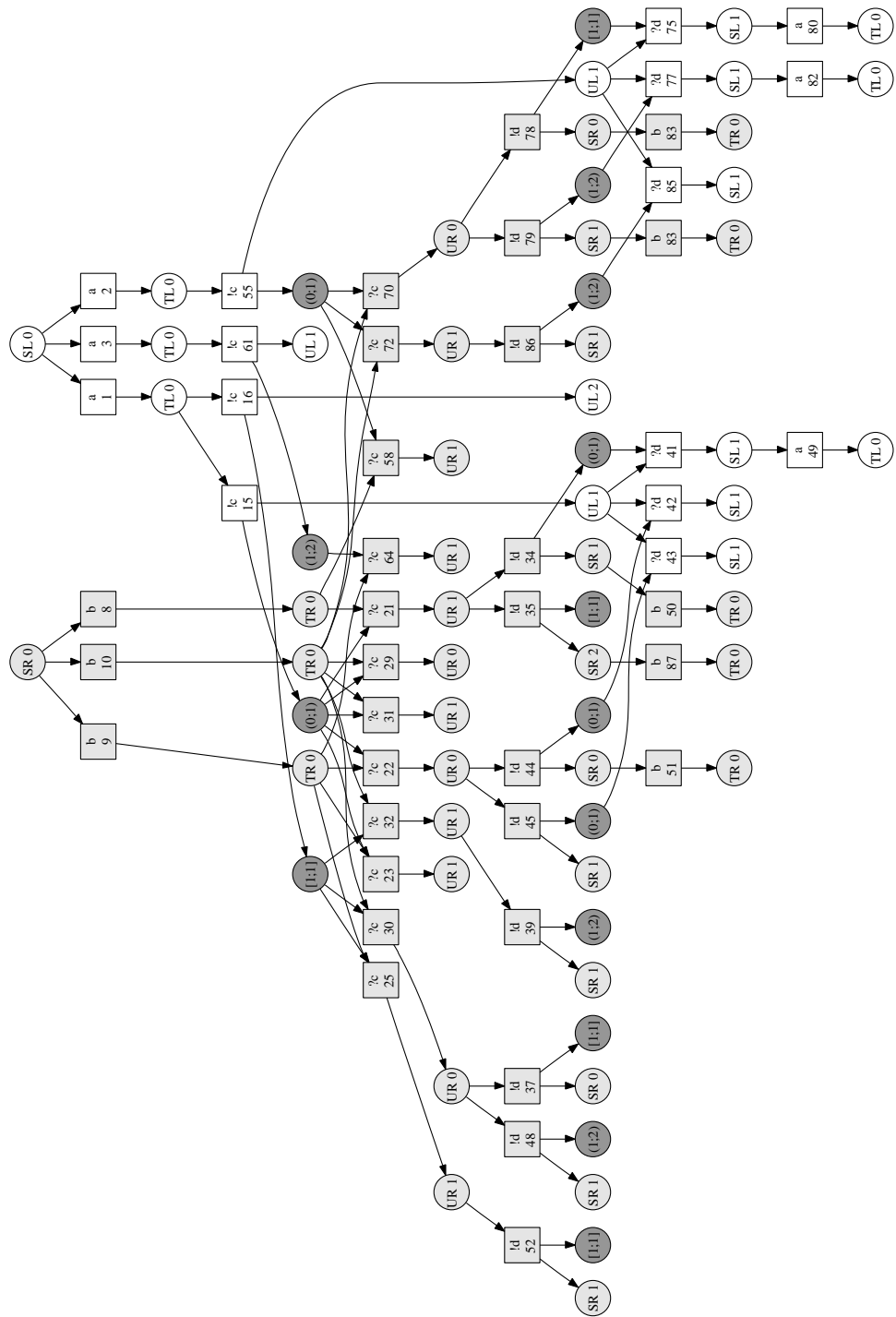


Figure 19: A folded Prefix of  $Unf(\mathcal{A}_3)$

clocks of different components are not important for an equivalent representation of an MTA's behaviour in the unfolding.

## 6 Summary, Conclusion and Future Work

**Summary** In this thesis we presented the theoretical foundations for a memory efficient verification technique for networks of timed automata. A network of timed automata is defined as a the parallel composition of several timed automata such that the automata can synchronize; the mode of synchronization we considered in this thesis is asynchronous message exchange. The technique aims at alleviating the state space problem due to concurrent actions in discrete, distributed, timed systems. The suggested technique uses a partial order semantics to succinctly represent the state space of such a network as a complete finite prefix of an unfolding.

The reason for a specific approach for timed, distributed systems is the inapplicability of the reduction techniques that were designed for untimed, distributed systems. The examples in the introduction show that, unlike in untimed systems, concurrently enabled actions in different components may not be independent in the presence of time. This simple, syntactically verifiable criterion, which is the basis for several state space reduction techniques in distributed, untimed systems, is not applicable in timed systems.

A principal approach for a successful reduction technique in the timed case, which we followed in this thesis, is to provide equivalently expressive semantics in which the independence of actions can be decided by a similarly simple criterion like in untimed systems. Then existing approaches for untimed systems are applicable to timed systems. The basic idea for such a semantics of networks of timed automata is to desynchronize the progress of time such that time may advance in each component independently. Synchronicity of time is required only whenever two components communicate. If two components do not synchronize, their (timed) behaviour cannot influence each other. We adapted this idea from Bengtsson et al. [BJLY98]. We analyzed in Sect. 1.3.1 the approach of Bengtsson et al. that proposes a state space reduction technique based on ample sets for networks of timed automata that synchronize on common actions. There we found that a successful reduction technique for networks of timed automata relies heavily on two aspects:

1. The time-abstract (and hence finite) representation of a state of the timed system needs to be partitionable into the states of its components.
2. The mode of synchronization in a network of timed automata in combination with a given independence relation on the system's actions influences whether the untimed language that is accepted by the network in the new semantics is regular. Regularity is a sufficient criterion for a finite representability of the reduced state space, which is necessary for an applicable technique.

We explained in Sect. 1.3.1 how both aspects did not get the necessary attention in the existing works of Bengtsson et al. [BJLY98] and Minea [Min99b], and in which way this influences a successful application of the technique.

We addressed the second aspect by choosing a specific class of networks of timed automata that synchronize on messages. We took the structural constraints of netcharts (see Sect. 1.3.3) and defined the class of *message passing timed automata* (MTA) in

Sect. 2. We defined their *global semantics* as an extension of the semantics of classic timed automata and then provided a step-wise refinement towards the *local-clock semantics* with desynchronized progress of time. We proved that local-clock semantics express exactly the same set of reachable states as the global semantics (Sect. 2.6).

We showed in Sect. 2.8 that MTA, just like timed automata, have a finite representation of their state space through a time-abstract representation of their states. We applied the region abstraction of Alur and Dill [AD94] which yields the *region automaton* of the system. The states of the region automaton contain *clock regions* that symbolically describe infinite sets of *clock valuations* over all clock variables of the system. Addressing the first aspect for a successful reduction technique, we defined a time-abstract representation of a MTA's state, the *decomposed region*, which consists of the structurally separated, local states of the system's components; this notion of a state has for each component a clock region over the clocks of this component only. Based on this representation, we proposed the time-abstract *local-region semantics* for MTA in Sect. 2.8 where the progress of time is desynchronized as well.

In the local-region semantics we showed that concurrently enabled actions in different components are independent (Sect. 2.9.1). With this property at hand, we defined a *partial-order semantics* for MTA by the help of a *labelled event structure*, which we call *unfolding*, see Sect. 2.9.2. We proved in Sect. 3 that this partial-order semantics yields a complete time-abstract representation of a MTA's state space: the unfolding of a MTA equivalently represents the sequential state space.

To ensure the applicability of the partial-order semantics for a state space reduction method according to the second aspect mentioned above, we showed that the unfolding of an MTA, where the guards of all actions have an upper bound, has a *complete, finite prefix* that contains the set of all reachable states of the system and all enabled actions; the proof is given in Sect. 4. We then informally proposed a technique to construct such a complete finite prefix of a MTA's unfolding, where we also highlighted open problems that need to be solved first in order to obtain an algorithmic solution.

**Conclusion** To evaluate the amount of reduction that can be achieved by representing the state space through a complete, finite prefix of an unfolding, we did a case study in Sect. 5. For a small example system, we compared its region automaton with a manually calculated prefix of its unfolding. We showed that for the example system, the space complexity of the prefix is still exponential in the size of the system description. A closer look reveals that there are different aspects that influence the size:

1. The unfolding *does* avoid the state space explosion due to concurrently enabled actions because they are not ordered. For  $n$  concurrently enabled actions in a state, the unfolding contains  $n$  events (in the best case) where the region automaton always requires  $2^n$  states.
2. The exponential size of a MTA's region automaton and of its prefix is, to a large extend, due to the presence of time in the state space: a finite representation of time still generates exponentially many clock regions and hence exponentially many different states in the number of clock variables in the system.

3. Exponentially many different states due to different clock regions have a worse effect on the unfolding compared to the region automaton.

The progress of time allows an action to occur at different moments, which may lead to different (conflicting) successor states due to the same action. This causes an exponential blow-up of the prefix in which a common successor of two conflicting states is represented twice – one successor for each state, while the region graph has no redundant representation of the same state and the growth due to conflicting states is less.

4. The construction of the unfolding and of the prefix requires the technical overhead of the *reference clocks*. Since size of the region automaton is exponential in the number of clocks in the system, the technical overhead may render the space complexity of an unfolding even worse than that of the region automaton.

Fortunately, the last three aspects can be influenced to yield better results and to estimate the applicability of the approach. The detailed look on the influence of the number of clock variables in the system, reference clocks and conflicting successor states in Sect. 5.4 allows us to draw the following conclusions.

1. For systems with a large degree of concurrency, i.e. many concurrently enabled actions, and few conflicting successor states due to the same action, the state space explosion in the region automaton by concurrent actions is worse than the exponential blow-up in the prefix due to conflicting successor states. For this class of systems, the prefix provides a more succinct representation of the state space than the region automaton.
2. The approach can be improved to reduce the blow-up due to concurrent successor states. We need the technical overhead of the reference clocks to determine the moments when two components may synchronize. Once these moments have been determined, the reference clocks are no longer necessary.

The example in Sect. 5.4 has shown that a reference clock may be the cause of conflicting successor states in the unfolding while the region automaton has no conflicting transitions for the same action. We found that we can fold the complete finite prefix by ignoring the reference clocks, thus limiting the growth due to conflicting successor states in the prefix. For the considered example, we achieved a prefix with less space requirements than the region automaton.

This exemplary additional reduction hints that a more careful approach in the construction of a complete, finite prefix may directly yield a smaller representation of the state space.

3. Region automaton and unfolding use a different notion of state. A state of the region automaton contains a single clock region over all clock variables of the system. The notion of state in the unfolding provides a clock region for each component, that considers the clocks of this component only.

If each component  $i = 1, \dots, P$  has  $n_i$  clock variables, then the region automaton allows  $\mathcal{O}((n_1 + \dots + n_P)! \cdot 2^{n_1 + \dots + n_P})$  different clock regions, and thus even more different states. The unfolding allows just  $\mathcal{O}((n_1)! \cdot 2^{n_1}) + \dots + \mathcal{O}((n_P)! \cdot 2^{n_P})$  different clock regions. Because the unfolding represents a state only implicitly as a combination of local states, it is sufficient to store  $\mathcal{O}((n_1)! \cdot 2^{n_1}) + \dots + \mathcal{O}((n_P)! \cdot 2^{n_P})$  different local states in the unfolding.

Assuming that  $n_i \approx n_j$  for all  $i, j = 1, \dots, P$ , the region automaton features in the worst case  $\mathcal{O}((p \cdot n_1)! \cdot 2^{p \cdot n_1})$  different clock regions, while the unfolding has  $\mathcal{O}(p \cdot (n_1! \cdot 2^{n_1}))$  different clock regions.

Thus the space complexity of the unfolding due to the representation of time in the state is significantly lower than the complexity in the region automaton.

These conclusions show that our approach can reduce the state space complexity due to concurrently enabled actions *and* due to the representation of continuous time in the state space. We thereby improved the results of Bengtsson et al. and Minea in the sense that our reduction technique exploits the independence relation to the full extend. We think that our approach is superior to the state space reduction techniques for timed automata that use ample sets. The latter techniques cannot reduce the exponential complexity that is induced by clock regions, since they can use the notion of a global state only. Despite these results, it needs to be noted that the prefix is still exponential in the size of the system description.

**Future Work** This thesis provided only the theoretical foundations for a reduction method. For an actual solution of the problem, the method needs to be implemented. The critical open work that needs to be done here is to find an algorithm that constructs the complete finite prefix. This requires further theoretical work regarding the cut-off criterion that allows to terminate the exploration of the state space without omitting a reachable state. It is desirable to find a cut-off criterion that is applicable to any configuration and not only to a restricted set of configurations (see Sect. 4.4 for details). The algorithm in combination with a suitable cut-off criterion should construct a *minimal* complete finite prefix of an MTA's unfolding.

Additionally, it would be desirable to spare the reference clocks in the construction of the prefix. Since the reference clocks are an integral part of the unfolding, it might not be possible to spare them completely. But it is likely that after some initial part of the prefix has been constructed, the reference clocks in this part play no further role in the construction of the remaining prefix. Then one could remove the reference clocks from this part and fold it, thus implementing a method that alleviates the exponential growth of the prefix due to conflicting successor states.

Another improvement of our method is to use *clock zones* instead of clock regions to symbolically represent infinite sets of clock valuations. The average case complexity of the zone representation outperforms the region representation, because a zone can be conceived as a convex union of regions [AD94]. Unfortunately, there is a small difficulty in using the zone abstraction, which we explained in Sect. 2.8.1: A clock zone may

contain two time points where a message that is sent at one time point can always be received, while a message that is sent at another time point can never be received. The finer grained region abstraction guarantees that these two time points are placed in different clock regions. This distinction is necessary to obtain a correct abstraction in the local semantics. Therefore, the open problem is to determine where a zone needs to be splitted into two zones such that all time points in a zone are equivalent wrt receivability of sent messages.

Finally, it is desirable to apply the method to arbitrary MTA. In further work, we need to examine whether the method is applicable in case the guards of an MTA can freely be chosen. Furthermore it would be interesting to see whether we could relax the structural constraints of netcharts which we applied to MTA and still apply our method.

Our claims on the amount of reduction obviously need to be validated experimentally in several case studies that compare especially the size of the region automaton and of the prefix in the average case.

## Index

- B*-bounded, 24
  - netchart, 14
- action
  - independent, 41
  - receive, 22
  - send, 22
- atomic clock constraints, 19
- buffer state function, 23
  - empty, 23, 26
  - region-stamped, 34
  - time-stamped, 26
- catch up, 48
- causality relation, 44
- clock formula, 19
- clock valuation, 20
- clock variables, 19
- concurrency relation, 44
- configuration, 44
  - prime configuration, 44
- conflict relation, 44
- dependency relation, 41
- enabled
  - action, 21
  - at time stamp, 22
- event, 44
- event structure, 15, 44
  - labelled, 15
- extension, 44
- independence relation, 41
- length of a run, 21
- linearization
  - of a configuration, 45
- location
  - of a clock variable, 22
  - of an action, 22
- MTA, 22
- netchart, 14
- reference clock, 25
- reference clock variable, 25
- region, 30
  - decomposed, 39
  - stamped, 34
  - successor, 31
  - synchronizable, 48
  - synchronized, 48
- region automaton, 31
  - local, 40
- region equivalence, 30
- region stamp, 33
- region-equivalent
  - transformed representation, 54
- run, 21
  - labelled, 21
  - local region
    - non-composed, 40
  - local-region, 40
  - timed, 21, 27
- state
  - discrete, 20
  - initial, 20
  - synchronizable, 29
  - synchronized, 29
  - timed, 20, 23
- step
  - combined, 20
  - composed, 24, 29
  - discrete, 20, 23, 31
    - local, 28
  - time, 20, 24, 31
    - local, 29
- synchronizable, 48
- synchronized, 48
- ta-sequence
  - local-region, 40
- time stamp, 21, 25, 26

- time-stamped action sequence, 27
- timed automaton, 19, 20
  - message passing, 22
  - progressing, 58
- trace, 43
  - prime, 43
- trace equivalence, 43
- transition system, 21
  - time-stamped, 27
- unfolding, 46

## List of Figures

1	Unordered Actions . . . . .	2
2	State Space Explosion . . . . .	4
3	Unfolding . . . . .	5
4	Commuting and Non-Commuting Actions in Timed Systems . . . . .	6
5	Dependent Actions in the Zone Automaton. . . . .	10
6	Dependent Actions in the Zone Automaton. . . . .	11
7	Network $N_5$ of Timed Automata $TA_1, TA_2$ with non-local choice. . . . .	13
8	A netchart of three components $A_1, A_2$ , and $A_3$ on the left and its equivalent representation as a Petri net on the right. . . . .	14
9	A prefix of the unfolding of the netchart in Fig. 8 in the graphical notation of a petri net. . . . .	16
10	Semantics of MTA and their relations. . . . .	19
11	A MTA where the receiving action $?a$ gets disabled before the sending action $!a$ . . . . .	38
12	A $B$ -bounded MTA where arbitrarily local actions create infinitely many different buffer states. . . . .	39
13	A progressing MTA $\mathcal{A}_3$ . . . . .	63
14	The region graph of $\mathcal{A}_3$ from Fig. 13 . . . . .	64
15	A diamond in $R(\mathcal{A}_3)$ because of concurrently enabled events $b$ and $?d$ . . . . .	65
16	The state space of $R(\mathcal{A})$ . The run created by the “good” ta-sequence (7) is highlighted in grey. . . . .	66
17	The region graph of $A_l$ in $\mathcal{A}_2^+$ from Fig. 13 . . . . .	68
18	A complete finite prefix of $Unf(\mathcal{A}_3)$ . . . . .	70
19	A folded Prefix of $Unf(\mathcal{A}_3)$ . . . . .	74

## References

- [ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, New York, 1990. IEEE.
- [AD90] R. Alur and D. Dill. Automata for modeling real-time systems. *Lecture Notes in Computer Science*, 443,:322 – 335, 1990.
- [AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [BD00] Béatrice Bérard and Catherine Dufourd. Timed automata and additive clock constraints. *Inf. Process. Lett.*, 75(1-2):1–7, 2000.
- [BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial Order Reductions for Timed Systems. In *International Conference on Concurrency Theory*, pages 485–500, 1998.
- [DGKK98] Dennis Dams, Rob Gerth, Bart Knaack, and Ruurd Kuiper. Partial-order Reduction Techniques for Real-time Model Checking. *Formal Asp. Comput.*, 10(5-6):469–482, 1998.
- [Die95] Volker Diekert. *The Book of Traces*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1995.
- [Dil89] D. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. *Lecture Notes in Computer Science*, 407:197–212, Jun 1989.
- [EMCGP99] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
- [ERV96] Javier Esparza, Stefan Romer, and Walter Vogler. An Improvement of McMillan’s Unfolding Algorithm. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 87–106, 1996.
- [Feh99] Ansgar Fehnker. Scheduling a steel plant with timed automata. *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA '99)*, 00:280, 1999.
- [FS02] Hans Fleischhack and Christian Stehno. Computing a Finite Prefix of a Time Petri Net. In *ICATPN '02: Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets*, pages 163–181, London, UK, 2002. Springer-Verlag.
- [HMKT99] J.G. Henriksen, M. Mukund, K. Narayan Kumar, and P.S. Thiagarajan. Towards a Theory of Regular MSC Languages. BRICS Report RS-99-52, University of Aarhus, 1999.

- [HMKT00] J.G. Henriksen, M. Mukund, K. Narayan Kumar, and P.S. Thiagarajan. Regular Collections of Message Sequence Charts. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'2000)*, number 1893 in Lecture Notes in Computer Science, Bratislava, Slovakia, 2000. Springer-Verlag.
- [HSL97] K. Havelund, A. Skou, K.G. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: an industrial case study using uppaal. *18th IEEE Real-Time Systems Symposium (RTSS '97)*, 00:2, 1997.
- [Lil98] Johan Lilius. Efficient state space search for time petri nets. In P. Jancar and M. Krtinsky, editors, *Proc. of MFCS Workshop on Concurrency Proc. of MFCS Workshop on Concurrency, Brno '98*, volume 18 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998.
- [McM92] Kenneth L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In G. v. Bochmann and D. K. Probst, editors, *CAV '92: Proceedings of the Fourth International Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 164–177. Springer-Verlag, June 1992.
- [MF76] P. Merlin and D.J. Faber. Recoverability of Communication Protocols—Implications of a Theoretical Study. *IEEE Transactions on Communications*, 24(9):1036–1043, Sept. 1976.
- [Min99a] Marius Minea. Partial Order Reduction for Model Checking of Timed Automata. In *CONCUR*, pages 431–446, 1999.
- [Min99b] Marius Minea. *Partial Order Reduction for Verification of Timed Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1999. CMU-CS-00-102.
- [MKT03] Madhavan Mukund, K. Narayan Kumar, and P. S. Thiagarajan. Netcharts: Bridging the gap between HMSCs and executable specifications. In *CONCUR*, pages 293–307, 2003.
- [NPW79] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri Nets, Event Structures and Domains. In *Semantics of Concurrent Computation*, pages 266–284, 1979.
- [Pag96] Florence Pagani. Partial Orders and Verification of Real-Time Systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems: 4th International Symposium Uppsala, Sweden*, volume 1135 of *Lecture Notes in Computer Science*, pages 327 – 346, Jun 1996.
- [Pag97] F. Pagani. *Ordres partiels pour la vrification de systmes temps rel (Partial Orders for Verification of Real-time Systems)*. PhD thesis, Centre dEtudes et de Recherches de Toulouse, September 1997.

- [Pel94] Doron Peled. Combining Partial Order Reductions with On-the-fly Model-Checking. In *CAV '94: Proceedings of the 6th International Conference on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 377–390, London, UK, 1994. Springer-Verlag.
- [Rei85] Wolfgang Reisig. *Systementwurf mit Netzen*. Springer, 1985.
- [SBM06] R. Ben Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *Concur'06*, 2006. <http://www-verimag.imag.fr/~maler/Papers/interleaving.pdf>.
- [Sta90] P. Starke. *Analyse von Petri-Netz Modellen*. Teubner, Stuttgart, 1990.
- [TY01] Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25 – 68, Jan 2001.
- [Val91] Antti Valmari. A stubborn attack on state explosion. In *CAV '90: Proceedings of the 2nd International Workshop on Computer Aided Verification*, pages 156–165, London, UK, 1991. Springer-Verlag.
- [YS97] T. Yoneda and B.-H. Schlingloff. Efficient Verification of Parallel Real-Time Systems. In *Formal Methods in System Design*, volume 11, pages 187–215, Aug 1997.

## Acknowledgements

This thesis has been written under interesting circumstances, and for more than half of the time it hasn't been certain that I would succeed.

Therefore I'd like to thank those who consciously or unconsciously helped me to finish this thesis. First of all, I'd like to thank Wolfgang Reisig for making me finally go to Singapore where the idea for this thesis was born and for giving me the freedom, support, and gentle pressure that I needed to finish it. I owe no less to P.S. Thiagarajan who proposed the topic to me, who gave me the freedom to work on it following my own way, and whose intuition helped me through the difficult passages and whenever it seemed that I had reached a dead end. Large parts of the definitions and proofs have found their formal shape due to the demanding enquiries of Shaofa Yang who was always interested in my problems and solutions. Similarly, the members of the group "Theory of Programming", especially Christian Stahl, Peter Massuthe, Andreas Glausch and Niels Lohmann, with whom I had helpful discussions in the late stages of my work, and Birgit Heene for keeping the administrative burden endurable at all times.

I owe much to Hannes, Patrick, Manfred, and Janet who gave me the necessary support while I was deeply stuck in the jungle of formalisms; to Christine who shared my experiences in a similar situation. I'd like to thank my family, who had to step back for a long time and yet had the right words in the right moments. They provided me with the safety that gave me the freedom to think about the problems of this thesis.

Very special thanks go to the "Studienstiftung des deutschen Volkes" and the "German Academic Exchange Service" without which this thesis wouldn't have been realized.



## **Erklärung**

Hiermit erkläre ich, die vorliegende Arbeit „Unfoldings for Message Passing Timed Automata“ selbstständig und ohne fremde Hilfe verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Weiterhin erkläre ich hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 14. Juli 2006