ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

DOCTORAL THESIS

The Class of Timed Automata with Quasi-Equal Clocks

Author:
Sergio Christian Herrera Salazar

Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

Dekan:
Prof. Dr. Oliver Paul

1. Gutachter:
Prof. Dr. Andreas Podelski

2. Gutachter:
Prof. Dr. Ernst-Rüdiger Olderog

Vorsitz:
Prof. Dr. Peter Thiemann

Beisitz:
Prof. Dr. Christoph Scholl

Tag der Disputation: 14.07.2017.

Printed and/or published with the support of the
German Academic Exchange Service.

"Go confidently in the direction of your dreams! Live the life you've imagined. As you simplify your life, the laws of the universe will be simpler."

Henry David Thoreau (1817 - 1862)

ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

Abstract

The Class of Timed Automata with Quasi-Equal Clocks

by Sergio Christian Herrera Salazar

In this thesis we present an approach for reducing the number of *quasi-equal* clocks in timed automata. Intuitively, two clocks are called quasi-equal if and only if their values only differ at time points in which those two clocks are reset.

Quasi-equal clocks are typically used in distributed real-time systems modelled with networks of timed automata. These networks, for instance, implement a time-triggered architecture which uses the *Time Division Multiple Access (TDMA)* scheme. This scheme provides system components with exclusive access to a shared medium whenever they generate events at predefined time points.

TDMA divides time into *cycles* and cycles into *slots*. Each component is allocated a unique slot whose length is measured by a quasi-equal clock. At the end of each cycle and before a new one begins, each component resets its quasi-equal clock. By the interleaving semantics of timed automata these resets occur one after the other, however, the same configuration is reached regardless of the order in which all these resets are executed.

Resetting quasi-equal clocks induces intermediate configurations. The size of a set of intermediate configurations is determined by all possible permutations of resets of the quasi-equal clocks. This size increases exponentially in the number of quasi-equal clocks reset at the end of each cycle. The verification effort induced by exploring these intermediate configurations with tools like *Uppaal* is a major obstacle for model checking industrial-size networks.

In this thesis we present an approach for reducing the number of quasi-equal clocks, and thereby eliminating those intermediate configurations. We apply this approach on networks with quasi-equal clocks where the resulting networks reflect all reachability properties of their respective original counterparts. Our experiments with nine industrial case studies show significant savings of verification costs for properties verified in transformed networks, as compared to the costs of verifying the same properties in the original networks. Since neither syntactic nor semantic assumptions for networks are required, our approach applies to **the whole class of timed automata with quasi-equal clocks**.

ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

Zusammenfassung

The Class of Timed Automata with Quasi-Equal Clocks

by Sergio Christian Herrera Salazar

In dieser Doktorarbeit wird ein neuer Ansatz zur Reduzierung von *quasi-equal* Uhren in Zeitautomaten vorgestellt. Zwei Uhren heißen quasi-equal, genau dann wenn sich diese Uhren in ihren Werten nur an Zeitpunkten unterscheiden, zu denen sie zurückgesetzt werden.

Quasi-equal Uhren werden typischerweise in verteilten Echtzeitsystemen verwendet, die durch Netzwerke von Zeitautomaten modelliert sind. Diese Netzwerke implementieren zum Beispiel eine *Time-Triggered Architecture*, die das *Time Division Multiple Access (TDMA)* Schema verwendet, um Systemkomponenten exklusiven Zugriff auf ein gemeinsam genutztes Medium bereit zu stellen, wann immer diese Systemkomponenten Ereignisse zu vordefinierten Zeitpunkten erzeugen.

TDMA unterteilt die Zeit in *Zyklen* und diese Zyklen wiederum in *Slots*. Jeder Systemkomponente wird einem eindeutigen Slot zugewiesen, dessen Länge durch eine quasi-equal Uhr gemessen wird. Am Ende eines Zyklus' setzt jede Komponente ihre quasi-equal Uhr zurück, bevor sie mit einem neuen Zyklus beginnt. Durch die Interleaving-Semantik von Zeitautomaten werden die Uhren nacheinander zurückgesetzt. Jedoch erreicht das System dieselbe Konfiguration, unabhängig von der Reihenfolge, in der alle diese Resets ausgeführt werden.

Zwischenkonfigurationen werden durch Resets von quasi-equal Uhren generiert. Die Größe der auftretenden Menge von Zwischenkonfigurationen wird durch alle möglichen Permutationen von Resets der quasi-equal Uhren bestimmt. Diese Größe wächst exponentiell mit der Anzahl der quasi-equal Uhren, die am Ende eines jeden Zyklus' zurückgesetzt werden. Durch Untersuchung der Zwischenkonfigurationen mit Tools wie *Uppaal* wird der Verifikationsaufwand erhöht, was ein schweres Hindernis für die Analyse von großen Netzwerken darstellt.

Diese Arbeit präsentiert ein Verfahren, welches Netzwerke so transformiert, dass die Anzahl der quasi-equal Uhren reduziert und dadurch Zwischenkonfigurationen entfernt werden, jedoch alle Erreichbarkeitseigenschaften der ursprünglichen Netzwerken erhalten bleiben. Außerdem zeigen die in dieser Arbeit vorgestellten neun industriellen Fallstudien, dass sich die Kosten, um Eigenschaften in den transformierten Netzwerken zu verifizieren, signifikant reduziert haben. Da weder syntaktische noch semantische Annahmen an die Netzwerke getroffen werden müssen, gilt der in dieser Arbeit vorgestellte Ansatz für die gesamte Klasse von Zeitautomaten mit quasi-equal Uhren.

Acknowledgements

First and foremost, I thank *Prof. Dr. Andreas Podelski* for giving me the opportunity to carry out this work under his supervision. This opportunity allowed to experience the scientifically enriching environment lived at the *Chair of Software Engineering* led by him. Once again I thank him for each teaching that helped me to improve my work. Definitely, without his support this work would have not been possible.

I thank *Dr. Bernd Westphal* for the many lessons he gave me on mathematical rigour, and on how to write precise and concise sentences. Thanks to Bernd I write (or at least hope so) better papers. I thank him as well for his patience, for sharing his ideas and knowledge with me, and for all those long discussions that we even continued on weekends. Definitively, without the support given by Bernd this work would have not been possible. He leaves an immense influence on my style of carrying out scientific work.

Many thanks to *Prof. Dr. Ernst-Rüdiger Olderog* for reviewing this thesis. It is an honor for me that one of the authors of my favorite book on real-time systems dedicates hours from his precious time to review my thesis. I thank as well *Prof. Dr. Peter Thiemann* and *Prof. Dr. Christoph Scholl* for their participation in the examination committee. Once again I specially thank Prof. Dr. Peter Thiemann for writing several recommendation letters for me, and for offering my first HIWI-Job.

I thank my sponsors from Mexico, *Consejo Nacional de Ciencia y Tecnología (CONACyT)*, and from Germany, *Deutscher Akademischer Austauschdienst (DAAD)*, for awarding me with the scholarship that supported this work. My gratitude and recognition to the highest quality of service of both institutions will last for the rest of my life. Definitively, without the financial support from these institutions this work would not have been carried out. Many thanks once again.

I want to thank my research colleagues: *Dr. Jochen Hoenicke*, *Dr. Sergio Feo-Arenis*, *Dr. Marco Muñiz*, *Dr. Stephan Arlt*, *Dr. Sergiy Bogomolov*, *Dr. Corina Mitrohin*, *Dr. Martin Wehrle*, *Dr. Matthias Heizmann*, *Dr. Daniel Dietsch*, *Dr. Jürgen Christ*, *Dr. Evren Ermis*, *Tanja Schindler*, *Marius Greitschus*, *Christian Schilling*, *Vincent Langenfeld* and *Alexander Nutz* for providing me with an excellent environment at the university, and for giving me suggestions on how to improve my work.

I want to thank the office management team, *Berit Brauer* and *Marlis Jost* for their immense help on administrative work, conference trips, letters of recommendations, etc. My gratitude and recognition to their kindness and high quality of service will last for the rest of my life.

Warm thanks go as well to *Ursula Epe*, programme coordinator of the *M.Sc. Computer Science*. During my long stay as student at the *University of Freiburg* I have experienced her kindness, willingness to help, and the highest quality of her service. She provided me with counseling, housing offers, letters of recommendations, etc. Definitively, my gratitude and recognition to her will last for the rest of my life.

I want to thank the friends I made during my studies: *Pedro Borromeo*, *Daniel Sánchez*, *Said Lobo*, *Alibek Kulzhabayev*, *Tri Atmoko*. Their friendship, counseling, help and company enriched my student years. I thank as well *Tânia Vieira* and *Anna Vieira* for their support, motivation, kindness, love and smiles during the write-up of this thesis. Lots of love to them.

Contents

List of Figures

List of Tables

Dedico este trabajo a mi familia, especialmente a mi madre Maria Salazar Garzón. *Ella ha sido mi fuente de inspiración para lograr mis metas.*

Chapter 1

Introduction

In distributed real-time systems that follow a *time-triggered architecture* [1] components access a global time base by means of a *local clock*. These components often follow a time division scheme, e.g. a *Time Division Multiple Access (TDMA)* scheme [2], to gain exclusive access to a shared medium whenever they generate events at predefined time points.

Modelling the local timing behaviour of components and the timing synchronisation constraints among them, is the natural way to model distributed real-time systems by networks of timed automata [3].

The problem that we address in this work is related to the verification overhead caused by using local clocks in networks of timed automata which, for instance, implement a time division scheme like the TDMA.

A TDMA scheme divides time into *cycles*, then each cycle is divided into *slots*, and each component is allocated a unique slot where it can generate local or communication events to other components. By limiting the activity of each component to its allocated slot whose length is measured by the local clock, exclusive access to a shared medium is guaranteed.

After the end of each cycle each component resets its local clock at the same point in time so that a new cycle starts. Logically, these resets do not consume time and all of them occur at once. However, by the interleaving semantics of networks of timed automata, each reset occurs one after the other. Nevertheless, the configuration resulting after all of these resets are carried out is independent of any possible permutation of them.

Clocks whose values only differ at time points in which those clocks are reset, are called *quasi-equal* [4]. Quasi-equality of clocks induces equivalence classes in timed automata. Note that the local clocks used in networks implementing the TDMA scheme adhere to the notion of quasi-equal clocks.

Resetting quasi-equal clocks induce intermediate configurations. The size of a set of intermediate configurations is determined by all possible permutations of resets of the quasi-equal clocks. This size increases exponentially in the number of quasi-equal clocks reset at the end of each cycle. The verification overhead induced by exploring these intermediate configurations with tools like *Uppaal* [5], is a major obstacle for *model checking* [6] industrial-size networks.

We consider that verification overhead unnecessary since typically these configurations only differ in the information of which clock has already been reset and which clock has not. Thus, efficiently *encoding* these differences saves us that overhead. With the aim of removing the unnecessary verification overhead, we reduce the number of *quasi-equal* clocks in timed automata.

The *quasi-equal clocks reduction approach* [4, 7, 8, 9] consists of transformations:

1. for replacing all clocks from an equivalence class of quasi-equal clocks by a representative clock. The resulting transformed network reflects all reachability properties of the original network,

2. for queries on networks where quasi-equal clocks occur so that they can be verified in the respective transformed networks. The cost of verifying properties is much lower in transformed networks than in their original counterparts with quasi-equal clocks.

The approach is provided as an automatic *source-to-source* transformation which yields models optimised for verification. In these models further techniques for the analysis of timed automata can directly be applied, and the modelling language of Uppaal can be supported. In this way we bridge the gap between efficiency and readability by enabling the *modelling engineer* to use and maintain more natural representations of a system for validation purposes. Our transformation frees the engineer from implementing optimisations for verification purposes. This is specially relevant in situations where the modelling engineer and the *verification engineer* are two different persons, and the former lacks of an educational background on verification methods. These situations occur often in small and medium sized enterprises from the high-tech industry in the context of projects of real-time systems [10].

We have mentioned before that verifying properties in transformed networks yields savings in verification costs as compared to the cost of verifying properties in original networks with quasi-equal clocks. In [9] we present a theoretical analysis that allows us to quantify and justify those savings. Moreover, we present space and time bounds that characterise a less expensive model checking effort in transformed networks wrt. the effort in original networks with quasi-equal clocks.

The cheaper model checking effort is justified by the following facts:

1. Encoding in transformed networks all intermediate configurations induced in original networks by executing all possible permutations of resets of quasi-equal clocks, leads to a drastic reduction in the number of configurations that Uppaal explores when verifying properties in transformed networks.

2. A reduction in the size of the *Difference Bound Matrices (DBMs)* that Uppaal uses to represent *zones* [11] is yielded by using only the representative clock of each equivalence class. This reduction leads to a more efficient representation, storage, access and manipulation of DBMs in memory.

A theoretical analysis in [9] allows us to conclude that the complexity of the model checking problem for networks of timed automata using quasi-equal clocks (in certain cases) is exponential in the number of equivalence classes of quasi-equal clocks, while for networks of timed automata in general is exponential in the number of clocks in those networks [6]. In this work we update that theoretical analysis since we introduce improvements in transformed networks that in our experiments yield up to 9% more savings in verification time than those savings presented in [9].

1.1 Contributions of This Thesis

This section provides a summary of the overall contributions of this thesis:

1. We present an approach which advances the state of the art in model checking for timed systems, by providing an automatic reduction of the number of quasi-equal clocks in any network of timed automata. Our approach yields transformed networks where all properties of the original networks are reflected. The cost of verifying properties is significantly lower in transformed networks than in their original versions with quasi-equal clocks. Our approach does not impose any syntactical assumption neither on networks nor on single automata, thus we are able to transform **the whole class of timed automata with quasi-equal clocks**.

2. By implementing our approach, we found out that the worst-case complexity of the model checking problem for networks of timed automata with quasi-equal clocks (in cases where those clocks are reset by *simple edges*, c.f. Definition 22) is exponential in the number of equivalence classes of quasi-equal clocks, while for networks in general is exponential in the number of clocks in those networks [6].

3. In networks of timed automata transformed with our approach the full query language of Uppaal is supported, hence, any property on an original network can be transformed and verified in the respective transformed network.

4. Our source-to-source transformation is automatically delivered by our tool *sAsEt* [12]. sAsEt implements distinct versions of our transformation in order to support several versions of Uppaal. This source-to-source transformation outputs a network of timed automata where further techniques for the analysis of networks can directly be applied, and where the rich modelling language of Uppaal can be supported.

5. We provide formulas to quantify the number of reachable configurations (in terms of zones) in transformed automata, and thereby know the number of configurations saved in those automata. For certain transformed automata our quantification is exact, and for the other automata we bound that number of reachable configurations. To the best of our knowledge, no research before ours has been devoted to predict the number of zones output by the verification analysis of Uppaal.

6. We apply our quasi-equal clocks reduction approach on nine industrial cases studies, and provide extensive experimental results which show significant savings in verification costs.

   We compare the savings in memory consumption and verification time delivered by the distinct versions of our transformation, and for some transformed versions of our industrial cases studies we are able to predict their exact size.

1.2 The Quasi-Equal Clocks Reduction Approach: An Iterative Development

We here briefly recall the iterative development of the quasi-equal clocks approach. Most of the content of this work is provided by the following publications, however, in this work we introduce improvements for the transformation of networks. With these improvements greater savings in verification time are obtained than by the transformation presented in [9].

1. ***Reducing Quasi-Equal Clocks in Networks of Timed Automata***.
   In [4] we propose an approach based on quasi-equality of clocks which reduces the verification time of so-called *local queries*, i.e. queries wrt. a single timed automaton, in networks of timed automata satisfying syntactical assumptions called *well-formedness*. In this approach we transform the case studies: [13, 14, 15].

2. ***Quasi-Equal Clock Reduction: More Networks, More Queries***.
   Although the technique introduced in [4] shows promising results for transformed networks, that technique has two severe drawbacks. Namely, it loses all the information from *unstable configurations*, i.e. configurations where quasi-equal clocks differ on their values, and it supports only local queries. In [7] we carry out an enhancement of the transformation algorithm from [4]. That is, we introduce: (1) a transformation to support the full query language of Uppaal in transformed networks, by summarizing unstable configurations into one dedicated location without losing information. By using this location we restate properties querying unstable configurations. (2) Weaker well-formedness assumptions than needed by the approach in [4] which allow us to transform the case studies: [16, 17, 18].

3. ***Quasi-Equal Clock Reduction: Eliminating Assumptions on Networks***.
   The approach from [7] still requires strong syntactic assumptions on networks in order to soundly apply the reduction of the number of clocks. In [8] we propose a transformation which does not require any syntactic assumption on networks, and does not constrain the support of properties of networks. We also introduce a strong distinction of edges that reset quasi-equal clocks in order to maximize savings wrt. verification costs of properties. We include three case studies: [19, 20, 21] which the algorithm in [7] does not transform.

4. ***The Model Checking Problem in Networks with Quasi-Equal Clocks***.
   In [9] we present a solid theoretical background that justifies the meaningful savings in verification costs that our approach delivers. In this work we propose formulas to quantify the number of reachable configurations in transformed networks, and thereby know the number of configurations saved in those networks. By implementing our approach we find out that a new complexity class characterises a less expensive model checking effort in transformed networks wrt. the effort in networks with quasi-equal clocks. Additionally, we introduce improvements to our transformation algorithm in order to maximize savings in number of configurations, and of verification time of properties. We eliminate all remaining semantic assumptions from [8] on networks with quasi-equal clocks.

1.3 Sources and Outline

Most of the content of this thesis has been presented in the following conferences:

1. ***Reducing Quasi-Equal Clocks in Networks of Timed Automata*** [4] at *FORMATS'12*.
2. ***Quasi-Equal Clock Reduction: More Networks, More Queries*** [7] at *TACAS'14*.
3. ***Quasi-Equal Clock Reduction: Eliminating Assumptions on Networks*** [8] at *HVC'15*.
4. ***The Model Checking Problem in Networks with Quasi-Equal Clocks*** [9] at *TIME'16*.

In Chapter 2 we discuss related work. In Chapter 3 we present preliminaries. In Chapter 4 we recall the notion of quasi-equal clocks. In Chapter 5 we present a transformation to reduce the number of quasi-equal clocks. In Chapter 6 we show our transformation correct. In Chapter 7 we present a theoretical analysis that justifies savings obtained by our approach. In Chapter 8 we present experimental results. We conclude in Chapter 9.

Chapter 2

Related Work

2.1 Introduction

In this chapter we discuss some of the different approaches related to our quasi-equal clocks reduction approach. Note that we may use *states* as a synonym for configurations of timed automata, and by *state-space* we mean the set of reachable states of those automata.

2.2 State-Space Reduction Techniques

Memory consumption in state-space exploration is one major factor that hampers the usage of model checking tools on industrial-size networks of timed automata, due to the vast information, e.g. clock values and control locations of networks, generated and stored during that exploration.

In this section we present a selection of techniques closely related to our quasi-equal clocks reduction approach. These techniques share our aim of reducing the state-space exploration when model checking timed systems.

2.2.1 Active and Equal Clocks Reduction

The approaches in [22, 23, 24] eliminate clocks by using static analysis over single timed automata, networks of timed automata and parametric timed automata, respectively. The approaches in [22, 23] reduce the number of clocks in timed automata by detecting *equal* and *active* clocks. Two clocks are equal in a location if both are reset by the same incoming edge, so just one clock for each set of equal clocks is necessary to determine the future behaviour of the system. A clock is active at a certain location if this clock appears in the invariant of that location, or in the guard of an outgoing edge of such a location, or another active clock takes its value through an outgoing edge. Non-active clocks play no role in the future evolution of the system and therefore can be eliminated.

Actually, active clock reduction is a special case of the more general location dependent and guard abstraction techniques of [25]. In [24] the same principle of active clocks is used in parametric timed automata.

Our industrial case studies use at most one clock per component which is always active. Moreover, the clocks in those case studies are quasi-equal, hence the equal and active approach will reduce no clocks in our industrial case studies. However, combining both approaches could increase the reduction of clocks in timed automata in two steps. In the first we eliminate equal and non-active clocks, then in a second step we eliminate quasi-equal clocks.

The work in [26] is another approach on reducing active clocks in timed automata, but in this approach the detection of active clocks is carried out on a semantic representation, i.e. the *zone graph*, of the timed automaton rather than on its syntactic form as the previous approaches do.

2.2.2 Reduction of Clocks by Abstraction of Systems

Global Clocks

The approach in [27] proposes a technique for reducing clocks in systems consisting of timed components. Each timed component represents a process and uses one internal clock for its internal operations. The timed components process input events, and trigger a *wave* of events along the system – in the form of a sequence of input-output events – with some delay bound within a given interval.

There is a working sequence for timed components. Each timed component is activated when it receives and process an input, i.e. the output from a previous component. Each timed component resets its internal clock at activation. Within this context, every internal clock is used only to ensure the time point that an output event needs to be produced. Therefore each component is only active during a small fraction of time to process the events generated along each wave. Because every internal clock is only measuring a small fraction of time during a wave, therefore, only one clock can be used to measure the duration of the complete wave.

The clocks in the industrial case studies we consider can in general not be reduced by this approach as components in those cases do not assume a working sequence of components.

Modelling multitasking applications running under real-time operating systems compliant with the *OSEK/VDX* standard, can be achieved with networks of timed automata. Each task is modelled by one timed automaton. The execution time of each task is measured by one internal clock.

The work in [28] proposes a technique to reduce clocks by constructing networks where only one clock measures the execution time of all tasks. Total o partial completion of each task is modelled by the expiration of its corresponding execution time. However, only one task can be running on the processor at a given time, and only the running task can be finished. Therefore, only the clock of the currently running task can affect the progress of the model. Hence, only one clock can be used for measuring the execution time of the currently running task.

The technique in [28] is specially crafted for multitasking applications running under real-time operating systems. Since none of our industrial case studies model those applications, therefore, that technique will not reduce clocks in our case studies.

Live Variable Analysis

The work in [29] presents a component *slicing* technique to shorten computation paths if those paths do not influence a property being verified. The work in [29] performs an analysis of variables, e.g. clocks, which occur in those paths candidates to be shortened. If those variables are irrelevant not only for enabling those paths but as well for the property being verified, then these variables are safely eliminated.

Applying this technique in our industrial case studies will show no reduction of clocks. Each component in our case studies uses at most one clock and each clock is relevant for the timing behaviour of the system.

The works in [30, 31] use *observers*, i.e. automata representing safety or liveness properties, to detect irrelevant modelling elements (clocks or even a whole automaton) that do not influence a property being verified. These irrelevant elements are deactivated (or ignored) without compromising the validity of an arbitrary property.

Constructing one observer per safety or liveness property in our case studies will unnecessarily increase their respective number of configurations to be explored, therefore we do not use this technique. However, case studies candidates for this technique and which use quasi-equal clocks can profit from our quasi-equal clocks reduction approach.

CEGAR

The approaches in [32, 33, 34] are three examples of the iterative technique *Counter-Example Guided Abstraction Refinement* (CEGAR) applied in the context of model checking for timed systems. Intuitively, to verify a network of timed automata against a safety property the CEGAR technique automatically constructs an abstract system from the concrete one. Depending on the implementation of the CEGAR the construction of the abstract system begins with, either some of the variables (clocks and data) of the concrete system or no variables at all.

Once that the first version of the abstract system is constructed, Uppaal is used to verify the property. If Uppaal reports that the abstract system satisfies the property, it can be concluded that the original system also satisfies the property, because the abstracted system is an over-approximation of the concrete one. If Uppaal finds a counterexample in the abstract system then an automatic analysis is carried out by a *counterexample analyser*, e.g. a test automaton or an algorithm, to determine whether the counterexample is spurious or not. If the analyser outputs that the counterexample is real, then it can be safely concluded that the property does not hold in the concrete system either. Otherwise, the initial abstraction was too coarse, and it must be in the next iteration refined by adding variables so that the spurious counterexample is eliminated.

These CEGAR techniques are in principle property oriented. For instance, if a concrete system contains five quasi-equal clocks and in a property four of them occur, the initial abstract system according to [32] will contain four quasi-equal clocks. In the cases [33, 34] the initial abstract system begins with zero clocks. However, that number increases in each iteration since the property for being satisfied may require four quasi-equal clocks.

In general, is not guaranteed that the number of clocks are reduced by these three approaches, while with our approach all clocks from an equivalence class are replaced by a representative clock. Our source-to-source transformation outputs a network of timed automata which is the input for the techniques here discussed. We can use CEGAR techniques on our transformed networks to reduce other elements of networks, e.g. data variables, locations, etc., that do not play a role for satisfying a given property.

2.2.3 Partial Order Reduction Techniques

Partial order reduction techniques are numerous, have been extensively investigated and applied in the context of timed automata. We discuss a selection of these techniques [29, 35, 36, 37, 38, 39, 40]. By these techniques a significant reduction of the reachable state-space of timed systems can be achieved when performing model checking.

Partial order reduction techniques exploit the permutability of concurrently executed transitions. That permutability yields the same state independently of the order of execution of those transitions. In timed automata these transitions are justified by edges which are *independent* [37]. These edges do not disable each other, and can be taken in any order and yet induce the same state.

By these techniques a model checking algorithm computes a representative permutation to be explored without changing the structure of the underlying states. In networks using quasi-equal clocks we have distinguished the edges that reset those clocks into *simple* and *complex* edges. Simple edges are independent by exclusively resetting quasi-equal clocks, and by not allowing communication neither through channels nor through data variables. In this way simple edges do not disable each other, and any permutation justified by these edges results in the same state.

The quasi-equal clocks reduction approach exploits the permutability of simple edges, and removes computational effort by proposing a fixed permutation of transitions justified by those edges (see [9]). In this work we replace that permutation of transitions by only one transition, similar to [29]. Furthermore, by reducing clocks our approach changes the state structure by making DBMs more compact. Compact DBMs lead to a more efficient representation, storage, access and manipulation of DBMs in memory. The opposite is done in [36] which for implementing a partial order reduction technique, clocks are introduced instead of being reduced.

The works in [38, 39, 40] address the problem that each clock in each system component advancing at the same rate may hinder the application of partial order reductions. For instance, the enabledness of a clock guarded transition c may depend on the order in which transitions a and b reset some clocks relevant for c. Hence, transitions a and b are not independent since their execution do not result in the same state, and only transition a or b enables c.

The works in [38, 39, 40] propose *local time semantics* to allow the implementation of standard partial order reduction techniques in cases like the example from above. Local time semantics allow each clock in each component to advance at its own rate, independently of the rate of other clocks in other components. Only when components synchronise on channels, local time rates of the participating automata are synchronised by new auxiliary clocks.

Assuming the *global time semantics* from [41], i.e. all clocks in the system advance at the same rate, does not hinder us to apply our particular instance of a partial order reduction technique, because: (1) we use the assumption that quasi-equal clocks have the same value except a time points where those clocks are reset and, (2) the order by which simple edges are taken leads to the same state.

Moreover, works in [38, 39, 40] rely on additional auxiliary clocks to implement partial order reduction techniques on the systems being studied. Hence, new clocks enlarge the state structure instead of reducing it. These works only support properties related to the representative permutation resulting from the partial order reduction technique applied. Networks transformed with our approach preserve all properties from the original networks, and in particular properties related to the states that our instance of a partial order reduction technique eliminates.

We conclude that partial order reduction techniques are complementary to our approach. Our approach is implemented as a source-to-source transformation. That transformation does not require any particular translation to implement the techniques here discussed, thereby, these techniques can be further applied to remove unnecessary interleavings not induced by simple edges.

2.2.4 Sequential and Layered Compositions

Uppaal-like tools typically perform a *parallel composition* operation (shown in Chapter 3) among all components of the system being model checked. This composition, for instance, in a system with disjoint activity introduces locations and edges that unnecessarily increase the state-space of the system. Thereby, the underlying effort to model check a system with disjoint activity increases as well.

In the following we discuss several works [37, 42, 43] that propose alternative compositions, e.g. *sequential* and *layered*, for system components in order to reduce the underlying state-space of their composition. We discuss the similarities of these works with the quasi-equal clocks reduction approach.

In [42] systems following a time division scheme are analysed, in particular those implementing the TDMA scheme. TDMA divides time into cycles and each cycle is divided into slots. Each component with its own clock is allocated a unique slot to perform its activity, e.g. sending of messages, transiting to locations, etc., almost exclusively. This division of time yields that the activity of each component is disjoint from each other's, that is, while one component is active the rest are idle except at the end of each cycle.

Evaluating the *enabledness* of most of the edges resulting from the parallel composition operation, is unnecessary in networks implementing the TDMA scheme given the disjoint activity assumption. This evaluation increases the cost of model checking those networks.

In order to obtain speedups of the model checking procedure for systems with disjoint activity, the work in [42] introduces a semantic sequential composition. This composition together with an overclock for all components, i.e. the representative of all components' clock, yields a transformed system with a reduced number of: clocks, locations and edges, than the one obtained with the parallel product.

This semantic concatenation transformation is complementary to our approach. We can apply in a first step our quasi-equal clocks reduction, and in a second step the semantic concatenation transformation in order to obtain extra speedups of the model checking procedure. No additional translation of our transformed networks is required for the second step.

Systems can be divided into *layers* consisting of actions distributed over the whole system [44]. The works in [37, 43] propose a layered composition for timed automaton which is intermediate between the parallel and sequential composition. The layered composition yields a reduction of the state-space through the exploitation of the so called *independence* between actions of a given system. Actions are independent if they do not disable each other, and if executing those actions in any order starting from a given state, results in the same state or in an equivalent one. Note that this notion of independence is common in partial order reduction techniques.

In a system consisting of two components where actions b of the second component have dependencies on the actions a of the first one, applying the parallel composition over these components does not respect the dependencies between a and b due to the *symmetric* nature of the composition. By symmetric nature we mean, synchronisation on common actions and interleaving on disjoint ones.
As opposed to parallel composition the layered composition over those components will respect those dependencies, i.e. in the layered composite system actions b are allowed to be executed only after actions a of the first component have been executed, while all other actions are not restricted. Thus, with regard to parallel composition, layered composition applies a partial ordering reduction technique.

Layered composition of timed automata is complementary to the quasi-equal clocks reduction approach, it does not change the structure of configurations, i.e. it does no reduce the number of clocks in the system, and can be applied in our case studies to obtain additional reductions of the state-space of systems.

2.3 Detecting Quasi-Equal Clocks

The notion of zones induces the *zone graph* whose construction for deciding reachability in Uppaal-like tools results feasible due to its size. This size although reduced in some cases can be infinite.

In principle, the zone graph can be used to detect quasi-equal clocks. In general the size of the zone graph defeats the goal of reducing the effort

induced by model checking transformed versions of timed automata with quasi-equal clocks. The output of this detection is an input that can be used for the transformation of those automata.

The work in [12] presents an abstraction of the zone graph called the *abstract zone graph*, which is coarse enough to yield a drastic reduction of the size of the zone graph, and precise enough to identify a large class of quasi-equal clocks. Intuitively, for constructing the abstract zone graph a coarse abstraction similar to the one yielded by the *widening operator* [23, 45] on zones, is applied to configurations induced by non-zero delays. During those delays no new quasi-equalities are neither introduced nor destroyed. In contrast, no proper abstraction is applied to configurations induced by a sequence of resets of quasi-equal clocks where time is not allowed to elapse. For detecting quasi-equal clocks is important to track those configurations where those clocks differ on their values.

An algorithm in [12] takes an automaton as input, constructs its abstract zone graph and traverses it on the fly. The abstract zones of that graph are used to check quasi-equalities for the clocks of the input automaton. The algorithm after traversing the whole graph outputs a set of equivalence classes of quasi-equal clocks. This algorithm is implemented in our tool sAsEt. We can use sAsEt to detect that set which is an input for our quasi-equal clocks reduction algorithm.

2.4 Quasi-Dependent Variables

The work in [46] generalises the notion of quasi-equality of clocks to *quasi-dependency* of variables in the context of hybrid automata. Thus, the notion of quasi-equality is a special case of quasi-dependency of variables.

In a similar way as for the quasi-equal clocks detection and reduction approach, the work in [46] performs in a first step an analysis to detect *quasi-dependent* variables, and in a second step a reduction of those variables in hybrid automata. Intuitively, two variables x and y are quasi-dependent if there exists a dependency function f that relates the values of x and y. For instance, in the case of quasi-equal clocks, say x and y, the dependency function f relates the values of those variables as follows, $f(x) = y$.

Since in hybrid automata resets of variables are not restricted to zero values, as for clocks in timed automata, values of quasi-dependent variables are not related by the underlying dependency function at most in *zero-time* configurations. In zero-time configurations time is forbidden to elapse. In those configurations quasi-dependent variables may assume any value without violating their quasi-dependency. In [46] a detection analysis similar to [12] establishes that the quasi-dependency function always holds in non-zero-time configurations.

The quasi-dependent variables detection and reduction approach in [46] replaces quasi-dependent variables by one representative variable and updates appropriately the system structure where those variables occur.
Properties of the original system are reflected, that is, a forbidden configuration is reachable in the transformed system if and only if it is reachable in the original system.

System complexity is reduced in two ways. First, there exists a reduction of the number of variables in the system. This step alone leads to a more efficient system handling as the underlying data structures become more compact. Furthermore, in many cases large performance boosts are achieved by completely avoiding interleavings of resets of quasi-dependent variables.

Actually, the quasi-dependent variables reduction can be only applied under severe syntactic assumptions on hybrid automata, similar to those that we have in [7] for timed automata. The results that we have obtained in the quasi-equal clocks reduction can be straightforward extended to the quasi-dependent variables reduction to transform more networks of hybrid automata with quasi-dependent variables. Interestingly, our future work goes in this direction.

2.5 Bounds of the Number of Zones

Predicting the number of zones in transformed networks helps us to know the size of those networks, and gives us an idea of the memory requirements needed to model check them.

To the best of our knowledge, no research before ours has been devoted to predict the number of zones output by the verification analysis of the tool Uppaal. Empirically, we have been able to propose formulas which in the best case exactly predict the number of zones of transformed networks, and in the worst-case bound that number of zones.
We believe that more exact formulas can be proposed as soon as the authors of the tool Uppaal publish its code, and researches understand how the tool calculates the number of zones output by the verification analysis.

Of course that zones can be broken down into clock regions [41] if the automaton is able to recognise them. The following publications propose bounds of the number of clock regions for the following flavours of timed automata: (classical) timed automata [3, 6, 41, 47], *event-clock automata* [48, 49], *updatable timed automata* [50, 51] and *probabilistic timed automata* [52].

2.6 Interleaving vs. Parallel Semantics

In Subsection 2.2.3 we discussed the permutability of concurrently executed transitions. That permutability may increase the size of the underlying state-space of timed systems. This permutability is one of the downsides of modelling timed automata with interleaving semantics.

In the work of [53] the new formalism called *finite state machines with time* (FSMT) is introduced. FSMTs use *parallel semantics* as an alternative for parallelising transitions which in timed automata are concurrently executed. *Semi-symbolic* approaches represent explicitly locations of timed automata and symbolically clocks values, e.g. zones. FSMTs work on *fully symbolic* state sets containing both the clock values and the state variables, e.g. locations in timed automata. These sets are technically implemented with *Linear And-Inverter-Graphs* (LinAIGs).

The work of [53] proposes two translations for networks of timed automata to FSMTs. One where the interleaving semantics are preserved, and another where each component of a FSMT can run in parallel, since the aim is to accelerate the model checking procedure for FSMTs without losing intermediate states induced by interleaving transitions. Thus, interleaving transitions are as well preserved even in the parallelised translation.

As opposed to [53] we reduce (quasi-equal) clocks, and thereby change the underlying state structure to make it more compact. In our work we parallelise (in one step) transitions justified by simple edges, and we do not offer both choices for simple edges, i.e. parallel and interleaving transitions.

In our work by parallelising transitions we do not risk losing intermediate states induced by interleaving transitions, since we transform properties querying these states. Hence, by our source-to-source transformation all properties are preserved. Moreover, in the prototype implementation of the formalism of [53], the prototype restricts assignments for integer variables to constants, variables and additions of two integers. Because of these restrictions the prototype could not transform our industrial case studies. Our source-to-source transformation supports the whole modelling and query language of Uppaal.

Chapter 3

Preliminaries

3.1 Timed Automaton

A timed automaton [41] is a finite automaton extended with clock variables ranging over real numbers. Those clock variables can be used to express constraints in transitions and in locations, called guards and invariants, respectively.
For self-containment the definitions in this section are borrowed from [3] (pages 136-145). Some of those definitions have been adapted to the context of this work.

Definition 1 *Clock Constraints.*

Let $\mathcal{X}$ be a set of clocks. The set $\Phi(\mathcal{X})$ of clock constraints over $\mathcal{X}$ is defined inductively by the following syntax: $\varphi ::= x \sim c \mid x - y \sim c \mid \varphi_1 \wedge \varphi_2$, where $\varphi \in \Phi(\mathcal{X})$, $x, y \in \mathcal{X}$, $c \in \mathbb{Q}_{\geq 0}$, and $\sim \in \{<, \leq, \geq, >\}$. $\diamondsuit$

Definition 2 *Timed Automaton.*

A (pure) timed automaton $\mathcal{A}$ is a structure $(L, B, \mathcal{X}, I, E, \ell_{ini})$ where:

- L, alternatively $L(\mathcal{A})$, is a finite set of *locations* with typical element ℓ.
- B, alternatively $B(A)$, is a finite set of *channels* with typical elements α, β. Note that $\alpha?$ denotes an *input* (or *listening*) and $\alpha!$ the corresponding *output* (or *sending*) on the channel $\alpha \in B$.
- $\mathcal{X}$, alternatively $\mathcal{X}(\mathcal{A})$, represents a finite set of *clocks*, with typical elements x, y.
- $I : L \to \Phi(\mathcal{X})$ is a function that maps to each location a clock constraint, i.e. its *invariant*.
- $E \subseteq L \times B_{?!} \times \Phi(\mathcal{X}) \times \mathcal{P}(\mathcal{X}) \times L$ is the set of *edges*, where $B_{?!} = \{a? \mid a \in B\} \cup \{a! \mid a \in B\} \cup \{\tau\}$ is the set of actions and $\tau \notin B$ is the *internal action* not visible from outside. $\mathcal{P}(\mathcal{X})$ denotes the set of reset operations. The alternative notation for E is $E(\mathcal{A})$. Each *edge* $e = (\ell, \alpha, \varphi, X, \ell') \in E$ describes an edge from (origin) location ℓ to (destination) location ℓ' labeled with the action α, the guard φ and the set X of clocks to be reset. We use $\varphi(e), \ell(e)$, etc. to denote the guard of e, the origin location of e, etc.
- $\ell_{ini} \in L$, alternatively $\ell_{ini}(\mathcal{A})$, is the initial location. $\diamondsuit$

Definition 3 *Valuations and Operations on Valuations*

A *valuation* ν of clocks in $\mathcal{X}$ is a mapping $\nu : \mathcal{X} \longrightarrow \mathit{Time}$ that assigns to each clock $x \in \mathcal{X}$ a time value. We write $\nu \models \varphi$, if ν *satisfies* the clock constraint φ, which is defined inductively:

$$\begin{array}{lll} \nu \models x \sim c & \text{iff} & \nu(x) \sim c \\ \nu \models x - y \sim c & \text{iff} & \nu(x) - \nu(y) \sim c \\ \nu \models \varphi_1 \wedge \varphi_2 & \text{iff} & \nu \models \varphi_1 \text{ and } \nu \models \varphi_2. \end{array}$$

Two operations involving *valuations* are:

- *Time-shift*. For a clock valuation ν for $\mathcal{X}$ and $t \in \mathit{Time}$ we write $\nu + t$ to denote the valuation with $(\nu + t)(x) = \nu(x) + t$ for all $x \in \mathcal{X}$.
- *Reset*. For a clock valuation ν for $\mathcal{X}$, a set $X \subseteq \mathcal{X}$ of clocks, and $t \in \mathit{Time}$ we write $\nu[X := t]$ to denote the valuation with

$$\nu[X := t](x) = \begin{cases} t, & \text{if } x \in X, \\ \nu(x), & \text{otherwise.} \end{cases}$$

$\diamondsuit$

Definition 4 *Semantics of Timed Automata.*

The semantics of a timed automaton $\mathcal{A} = (L, B, \mathcal{X}, I, E, \ell_{ini})$ is given by the (labelled) transition system:

$$\mathcal{T}(\mathcal{A}) = (\mathit{Conf}(\mathcal{A}), \mathit{Time} \cup B_{?!}, \{\xrightarrow{\lambda} | \ \lambda \in \mathit{Time} \cup B_{?!}\}, \mathcal{C}_{ini}),$$

where the following holds:

- $\mathit{Conf}(\mathcal{A}) = \{\langle \ell, \nu \rangle \mid \ell \in L \wedge \nu : \mathcal{X} \longrightarrow \mathit{Time} \wedge \nu \models I(\ell)\}$ is the set of configurations of $\mathcal{A}$.
- The set $\mathit{Time} \cup B_{?!}$ contains all labels that may appear at transitions.
- For each $\lambda \in \mathit{Time} \cup B_{?!}$ the transition relation $\xrightarrow{\lambda} \subseteq \mathit{Conf}(\mathcal{A}) \times \mathit{Conf}(\mathcal{A})$ has one of the following two types:
  - *Delay transition*, expresses that some time $t \in \mathit{Time}$ elapses but the location is left unchanged, i.e. $\langle \ell, \nu \rangle \xrightarrow{t} \langle \ell, \nu + t \rangle$ iff $\nu + t' \models I(\ell)$ holds for all $t' \in [0, t]$.
  - *Action transition*, expresses that an action $\alpha \in B_{?!}$ occurs and some clocks may be reset, but the time is unchanged, i.e. $\langle \ell, \nu \rangle \xrightarrow{\alpha} \langle \ell', \nu' \rangle$ iff there is an edge $(\ell, \alpha, \varphi, X, \ell') \in E$, where $\nu \models \varphi, \nu' = \nu[X := 0]$ and $\nu' \models I(\ell')$.
- $\mathcal{C}_{ini} = \{\langle \ell_{ini}, \nu_{ini} \rangle\} \cap \mathit{Conf}(\mathcal{A})$ with $\nu_{ini}(x) = 0$ for all clocks $x \in \mathcal{X}$ is the set of initial configurations. $\diamondsuit$

In this work for a configuration $s = \langle \ell, \nu \rangle \in \mathit{Conf}(\mathcal{A})$ we use ℓ_s to denote the location ℓ of s, and ν_s to denote the valuation ν of s. Alternatively, we may write $\langle \ell_s, \nu_s \rangle$ to denote s. Furthermore we write $s \models \varphi$ if and only if $\nu_s \models \varphi$ as previously defined.

Definition 5 *Reachability of Configurations.*

Let $\mathcal{A}$ be a timed automaton and $\mathcal{T}(\mathcal{A}) = (Conf(\mathcal{A}), Time \cup B_{?!}, \{\xrightarrow{\lambda} | \ \lambda \in Time \cup B_{?!}\}, \mathcal{C}_{ini})$ its transition system. A configuration $\langle \ell, \nu \rangle \in Conf(\mathcal{A})$ is *reachable* if and only if there is a sequence of the form:

$$\langle \ell_0, \nu_0 \rangle \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_n} \langle \ell, \nu \rangle.$$

With $\langle \ell_0, \nu_0 \rangle \in \mathcal{C}_{ini}$. If $\mathcal{C}_{ini} = \emptyset$ such a sequence does not exist. A location ℓ is *reachable* if and only if a configuration of the form $\langle \ell, \nu \rangle$ is reachable. The set $Reach_{\mathcal{A}}$ contains all reachable configurations of automaton $\mathcal{A}$, in other words $Reach_{\mathcal{A}}$ denotes the state-space of $\mathcal{A}$. $\diamondsuit$

Definition 6 *Computation Path.*

A *computation path (or simply a path) of $\mathcal{A}$ starting in the configuration* $\langle \ell_0, \nu_0 \rangle, t_0$, where the time stamp $t_0 \in Time$ corresponds to the value of a special clock that is never reset, is a sequence

$$\pi : \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \langle \ell_2, \nu_2 \rangle, t_2 \xrightarrow{\lambda_3} \ldots$$

of time-stamped configurations of $\mathcal{A}$ which is either infinite or maximally finite, i.e. the sequence cannot be extended any further by some time-stamped transition. A *computation path of $\mathcal{A}$* is a computation path starting in $\langle \ell_0, \nu_0 \rangle, 0$, where $\langle \ell_0, \nu_0 \rangle \in \mathcal{C}_{ini}$. We denote with π_i the ith element of a path π. The set of all paths of $\mathcal{A}$ is denoted by $Paths(\mathcal{A})$. $\diamondsuit$

Definition 7 *Real-Time Sequence.*

A *real-time sequence* is an infinite sequence $t_0, t_1, t_2, t_3, \ldots$ of values $t_i \in Time$ for $i \in \mathbb{N}$ with the following properties:

1. *Monotonicity:* $\forall\, i \in \mathbb{N} \bullet t_i \leq t_{i+1}$.
2. *Non-Zeno behaviour* or *unboundedness*: $\forall\, t \in Time\ \exists\, i \in \mathbb{N} \bullet t < t_i$. $\diamondsuit$

3.2 Networks of Timed Automata

A network of timed automata is obtained by using the parallel composition operator over single timed automata. In a network every single automaton executes transitions independently, or can communicate with other automata by a synchronized communication via *handshake*, by using complementary actions $a!$ and $a?$. For self-containment the definitions in this section are borrowed from [3] (pages 145-150, 166-167), unless other sources mentioned. Some of those definitions have been extended and adapted to the context of this work.

Definition 8 *Parallel composition.*

The parallel composition operator $\mathcal{A}_1 \,||\, \mathcal{A}_2$ of two timed automata

$$\mathcal{A}_i = (L_i, B_i, \mathcal{X}_i, I_i, E_i, \ell_{ini,i}),$$

$i = 1, 2$, with disjoint sets of clocks $\mathcal{X}_1$ and $\mathcal{X}_2$, and disjoint sets of locations L_1 and L_2, yields the timed automaton

$$\mathcal{A}_1 \,||\, \mathcal{A}_2 \stackrel{\text{def}}{=} (L_1 \times L_2, B_1 \cup B_2, \mathcal{X}_1 \cup \mathcal{X}_2, I, E, (\ell_{ini,1}, \ell_{ini,2}))$$

where the following holds:

- Conjunction of location invariants: $I(\ell_1, \ell_2) \overset{def}{\Longleftrightarrow} I_1(\ell_1) \wedge I_2(\ell_2)$.
- The transition relation E is constructed following these rules:
  - *Handshake communication*: τ the internal action is yield by synchronizing $a!$ with $a?$, i.e. if $(\ell_1, a!, \varphi_1, \mathcal{X}_1, \ell_1') \in E_1$ and $(\ell_2, a?, \varphi_2, \mathcal{X}_2, \ell_2') \in E_2$, or if $(\ell_2, a!, \varphi_2, \mathcal{X}_2, \ell_2') \in E_2$ and $(\ell_1, a?, \varphi_1, \mathcal{X}_1, \ell_1') \in E_1$, then also
  $$((\ell_1, \ell_2), \tau, \varphi_1 \wedge \varphi_2, X_1 \cup X_2, (\ell_1', \ell_2')) \in E.$$
  - *Asynchrony*: if $(\ell_1, \alpha, \varphi_1, \mathcal{X}_1, \ell_1') \in E_1$ then for all $\ell_2 \in L_2$ also
  $$((\ell_1, \ell_2), \alpha, \varphi_1, \mathcal{X}_1, (\ell_1', \ell_2)) \in E$$
  and conversely, if $(\ell_2, \alpha, \varphi_2, \mathcal{X}_2, \ell_2') \in E_2$ then for all $\ell_1 \in L_1$ also
  $((\ell_1, \ell_2), \alpha, \varphi_2, \mathcal{X}_2, (\ell_1, \ell_2')) \in E$. ♢

Definition 9 *Restriction.*
A local channel b is introduced by the restriction operator $chan\ b \bullet \mathcal{A}$ which, for a timed automaton $\mathcal{A} = (L, B, \mathcal{X}, I, E, \ell_{ini})$ yields the timed automaton

$$chan\ b \bullet \mathcal{A} \overset{\text{def}}{=} (L, B \setminus \{b\}, \mathcal{X}, I, E', \ell_{ini}),$$

where the following holds:

- Restriction: if $(\ell, \alpha, \varphi, X, \ell') \in E$ and $\alpha \notin \{b!, b?\}$ then $(\ell, \alpha, \varphi, X, \ell') \in E'$.

For list of channels we introduce the abbreviation

$$chan\ b_1 \ldots b_m \bullet \mathcal{A} \overset{\text{def}}{=} chan\ b_1 \bullet \ldots chan\ b_m \bullet \mathcal{A}.$$

♢

Definition 10 *Closed Network.*
A network $\mathcal{N} = chan\ b_1, \ldots, b_m \bullet (\mathcal{A}_1 \,||\, \cdots \,||\, \mathcal{A}_n)$ is called *closed* if all channels of the automata are local, i.e. if $\{b_1, \ldots, b_m\}$ is the set of all channels used in one of the $\mathcal{A}_i$. We use $\mathcal{A}_i \in \mathcal{N}$ to denote the ith automaton $\mathcal{A}_i$ of $\mathcal{N}$. We may omit the subscript if it is clear from the context. ♢

According to [3] in $\mathcal{N}$ each component automaton $\mathcal{A}_i$ has its own control location ℓ_i. Hence, for the whole network a *control vector* $\vec{\ell} = (\ell_1, \ldots, \ell_n)$ is a collection of the control locations of the components in that network. A change of the ith component's location from ℓ_i to ℓ_i' is denoted by $\vec{\ell}[\ell_i := \ell_i']$.

Definition 11 *Extended Clock Constraints.*
Let V be a set of data variables, with typical element v.

- The set $\Psi(V)$ of *integer expressions over* V, with typical element ψ_{int}, is defined by the usual syntax, using variables in V and the operator symbols $+, -, \ldots$.

- The set $\Phi(V)$ of *integer constraints or data constraints over* V, with typical element φ_{int}, is defined as the set of Boolean expressions with the usual syntax, using variables in V, operator symbols $+, -, \ldots$ and the predicate symbols $<, \leq, =, \geq, >$.

- Let $\mathcal{X}$ be a set of clock variables, with typical elements x, y. The set $\Phi(\mathcal{X}, V)$ of extended clock constraints with typical element φ is defined by the following syntax: $\varphi ::= \varphi_{clk} \mid \varphi_{int} \mid \varphi_1 \wedge \varphi_2$. Where $\varphi_{clk} \in \Phi(\mathcal{X})$ is a clock constraint, and $\varphi_{int} \in \Phi(V)$ is an integer constraint. $\diamondsuit$

Definition 12 *Extended Valuations and Operations on Extended Valuations.*
Valuations ν now assign values to both clocks and data variables. The satisfaction relation $\nu \models \varphi$ between valuations and guards extends Definition 3 for data constraints in the straightforward way.
For the extended definition of valuations we adapt the operations of time shift and modification.

- *Time-shift*. For a clock $x \in \mathcal{X}$, a data variable $v \in V$ and $t \in Time$, we write $\nu + t$ to denote the valuation with

$$\begin{array}{lcl} (\nu + t)(x) & = & \nu(x) + t, \\ (\nu + t)(v) & = & \nu(v). \end{array}$$

  A *modification* or *reset operation* is an assignment to a clock $x \in \mathcal{X}$, i.e. $x := 0$, or an assignment to a data variable $v \in V$ of the form, $v := \psi_i$, where $\psi_i \in \Psi(V)$.

- *Reset*. Let $\mathcal{R}(\mathcal{X}, \mathcal{V})$ denote the set of these reset operations, with typical element r. The modification of a valuation ν under a reset operation r is denoted by $\nu[r]$ and defined as follows:

$$\nu[x := 0](v') = \begin{cases} 0, & \text{if } v' = x, \\ \nu(v'), & \text{otherwise}, \end{cases}$$
$$\nu[v := \psi_i](v') = \begin{cases} \nu(\psi_i), & \text{if } v' = v, \\ \nu(v'), & \text{otherwise}. \end{cases}$$

By $\vec{r}$ we denote a finite list of reset operations on clocks and data variables, $\vec{r} = \langle r_1, \ldots, r_n \rangle$, and we extend the definition of modification appropriately:

$$\nu[\langle r_1, \ldots, r_n \rangle] = \nu[r_1] \ldots [r_n].$$

We use $\mathcal{R}(\mathcal{X}, \mathcal{V})^*$ to denote the set of these list operations and $\langle\rangle$ to denote the empty list of reset operations. $\diamondsuit$

According to [3] *Uppaal* restricts locations invariants to conjunction of constraints

$$x \preceq n \text{ with } \preceq \in \{<, \leq\} \text{ and } n \in \mathbb{N}.$$

With this restriction, location invariants $I(\ell)$ are *downward closed*, i.e. whenever $\nu + t \models I(\ell)$ then also $\nu + t' \models I(\ell)$ for all $t' \in [0, t]$.

Uppaal uses the following notion of extended timed automaton which we take from [3] and we here extend it by adding committed and urgent locations; rendez-vous, broadcast and urgent channels, and priorities on those channels.

Definition 13 *Extended Timed Automaton.*

An *extended timed automaton* $\mathcal{A}_e$ is a structure

$$\mathcal{A}_e = (L, B, \mathcal{X}, V, I, E, \ell_{ini})$$

where $\mathcal{X}, I, \ell_{ini}$ are defined as in Definition 2 of pure timed automata (but I is restricted as just explained) and where:

- L is the set of locations which contains $L^c, L^u \subseteq L$, i.e. the sets of *committed locations* ($L^c(\mathcal{A}_e)$ for short) and *urgent locations* ($L^u(\mathcal{A}_e)$), respectively.
- B is the set of channels where:
  - $B^r \subseteq B$ is the set of *rendez-vous channels* which model binary synchronization. Alternative notation $B^r(\mathcal{A}_e)$.
  - $B^b \subseteq B$ is the set of *broadcast channels* which model one-to-many synchronization. Alternative notation $B^b(\mathcal{A}_e)$.
  - $B^u \subseteq B^r \uplus B^b$ is the set of *urgent channels* which model "as soon as possible" synchronization, i.e. synchronization on urgent channels cannot be delayed. Alternative notation $B^u(\mathcal{A}_e)$.
- V, alternatively $V(\mathcal{A}_e)$, is a set of *data variables*, with element v.
- $E \subseteq L \times B_{?!} \times \Phi(\mathcal{X}, V) \times \mathcal{R}(\mathcal{X}, V)^* \times L$ represents the finite set of *edges*. Alternative notation $E(\mathcal{A}_e)$. Each *edge* $(\ell, \alpha, \varphi, \vec{r}, \ell') \in E$ that describes an edge from (origin) location ℓ to (destination) location ℓ' labeled with an action α, a guard φ, and a list $\vec{r}$ of reset operations.
- If $(\ell, \alpha, \varphi, \vec{r}, \ell') \in E$ and in α occurs a channel in B^u then $\varphi = true$. This condition prevents that urgent actions are prohibited by guards.

Referring to Definition 2 for I means that it assigns to each location ℓ an invariant $I(\ell) \in \Phi(\mathcal{X}) = \Phi(\mathcal{X}, \emptyset)$. Thus location invariants constrain only clocks but not data variables. ♢

The following functions *clocks* and *vars* allow us to obtain the clocks and the variables which are updated by a given edge. These functions together with the definition of *enabled edge* are not provided by [3].

Definition 14 *Functions clocks and vars.*
Let $\mathcal{A}$ be an extended timed automaton. Let $(\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{A})$ be an edge. Then functions *clocks* and *vars* retrieve each clock and each variable being updated in $\vec{r}$, respectively.

$$clocks(\vec{r}) = \begin{cases} \{x_1, \ldots, x_n\}, & \text{if } \vec{r} = \langle \ldots, x_1 := 0, \ldots, x_n := 0, \ldots \rangle, \\ & \text{and } x_1, \ldots, x_n \in \mathcal{X}(\mathcal{A}), \\ \emptyset, & \text{otherwise.} \end{cases}$$

$$vars(\vec{r}) = \begin{cases} \{v_1, \ldots, v_n\}, & \text{if } \vec{r} = \langle \ldots, v_1 := \psi_1, \ldots, v_n := \psi_n, \ldots \rangle, \\ & \text{and } v_1, \ldots, v_n \in V(\mathcal{A}), \\ \emptyset, & \text{otherwise.} \end{cases}$$

♢

Definition 15 *Enabled Edge.*
Let $e = (\ell, \alpha, \varphi, X, \ell') \in E(\mathcal{A})$ be an edge of a timed automaton $\mathcal{A}$. Let $\langle \ell, \nu \rangle \in Conf(\mathcal{A})$ be a reachable configuration of $\mathcal{A}$. Then we call e *enabled* if and only if the valuation ν satisfies φ, i.e. if $\nu \models \varphi$, and the valuation yielded by resetting the clocks in X satisfies the invariant of location ℓ', i.e. if $\nu[X := 0] \models I(\ell')$. ♢

In the following we introduce the semantics for extended timed automata which we take from [3] and here are extended by adding semantics for: rendez-vous, broadcast and urgent channels, and priorities on those channels. In this definition we use the function *chan* on actions, which yields the channel used in an input or output action.

Definition 16 *Semantics of Extended Timed Automata.*
Let $\rho : B \cup \{\tau\} \to \mathbb{N}$ be a function that assigns a *priority* to τ and to each channel in B. For extended timed automata $\mathcal{A}_{e_i} = (L_i, B_i, \mathcal{X}_i, V_i, I_i, E_i, \ell_{ini,i})$ with $i = 1, \ldots, n$, pairwise disjoint set $\mathcal{X}_i$ of clocks, and pairwise disjoint set L_i of locations, consider the closed network $\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$. Then its operational semantics is defined by the labelled transition system

$$\mathcal{T}(\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n)) = (Conf(\mathcal{N}), Time \cup \{\tau\}, \{\xrightarrow{\lambda} | \ \lambda \in Time \cup \{\tau\}\}, \mathcal{C}_{ini})$$

where:

- $\mathcal{X} = \bigcup_{k=1}^{n} \mathcal{X}_k$ and $V = \bigcup_{k=1}^{n} V_k$.
- $Conf(\mathcal{N}) = \{\langle \vec{\ell}, \nu \rangle \mid \ell_i \in L_i \ \wedge \ \nu : \mathcal{X} \cup V \longrightarrow Time \cup \mathbb{Z} \ \wedge \ \nu \models \bigwedge_{k=1}^{n} I_k(\ell_k)\}$ is the set of configurations of $\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$.
- For all $\lambda \in Time \cup \{\tau\}$ the transition relation $\xrightarrow{\lambda} \subseteq Conf(\mathcal{N}) \times Conf(\mathcal{N})$ has one of the following four types:
  - *Internal transition* $\langle \vec{\ell}, \nu \rangle \xrightarrow{\tau} \langle \vec{\ell}', \nu' \rangle$ occurs if for some $i \in \{1, \ldots, n\}$ there is a τ-edge $e = (\ell_i, \tau, \varphi, \vec{r}, \ell_i') \in E_i$ in the ith automaton such that:
    - $\nu \models \varphi$, i.e. the guard is satisfied.
    - $\vec{\ell}' = \vec{\ell}[\ell_i := \ell_i']$.

  - $\nu' = \nu[\vec{r}]$ and $\nu' \models I_i(\ell'_i)$.
  - If $\ell_k \in L^c_k$ for some $k \in \{1, \dots, n\}$ then $\ell_i \in L^c_i$, i.e. if there is a committed location in $\vec{\ell}$ then the ith automaton is in such a location.
  - If for all $j \in \{1, \dots, n\}$ such that $j \neq i$, there is $\bar{e} \in E_j$ such that the jth automaton is in $\ell(\bar{e})$ then $\rho(\tau) > \rho(chan(\alpha(\bar{e})))$, i.e. the internal transition induced by taking edge e has a higher priority over any transition induced by taking $\bar{e}$.

- *Synchronization transition* $\langle \vec{\ell}, \nu \rangle \xrightarrow{\tau} \langle \vec{\ell'}, \nu' \rangle$ occurs if for some $i, j \in \{1, \dots, n\}$ with $i \neq j$ and some channel $b \in B_i \cap B_j$ there are edges $e_i = (\ell_i, b!, \varphi_i, \vec{r}_i, \ell'_i) \in E_i$ and $e_j = (\ell_j, b?, \varphi_j, \vec{r}_j, \ell'_j) \in E_j$, i.e. the ith and the jth automaton can synchronize their output and input on the channel b, such that
  - $\nu \models \varphi_i \wedge \varphi_j$, i.e. both guards are satisfied.
  - $\vec{\ell'} = \vec{\ell}[\ell_i := \ell'_i][\ell_j := \ell'_j]$.
  - $\nu' = \nu[\vec{r}_i][\vec{r}_j]$ and $\nu' \models I_i(\ell'_i) \wedge I_j(\ell'_j)$.
  - If $\ell_k \in L^c_k$ for some $k \in \{1, \dots, n\}$ then $\ell_i \in L^c_i$ or $\ell_j \in L^c_j$, i.e. if there is a committed location in $\vec{\ell}$ then the ith or the jth automaton is in such a location.
  - If for all $q \in \{1, \dots, n\}$ such that $i \neq q \neq j$, there exists $\bar{e} \in E_q$ such that the qth automaton is in $\ell(\bar{e})$ then $\rho(b) > \rho(chan(\alpha(\bar{e})))$, i.e. the synchronization transition induced by taking edges e_i and e_j has a higher priority over any transition induced by taking edge $\bar{e}$.

- *Delay transition* $\langle \vec{\ell}, \nu \rangle \xrightarrow{t} \langle \vec{\ell}, \nu + t \rangle$ occurs if
  - $\nu + t \models \bigwedge_{k=1}^{n} I_k(\ell_k)$ holds, i.e. all invariants are satisfied at the end of the delay.
  - There is no $i \in \{1, \dots, n\}$ with $\ell_i \in L^c_i \cup L^u_i$, i.e. no automaton is in a committed or in an urgent location.
  - There are $i, j \in \{1, \dots, n\}$ such that there are edges $e_i = (\ell_i, b!, \varphi_i, \vec{r}_i, \ell'_i) \in E_i$ and $e_j = (\ell_j, b?, \varphi_j, \vec{r}_j, \ell'_j) \in E_j$, where the ith automaton is in $\ell(\bar{e}_i)$ and the jth automaton is in $\ell(\bar{e}_j)$ and $b \in B^u \cap B^r$ and $\nu + t' \not\models \varphi(e_i) \wedge \varphi(e_j)$ for all $t' \in [0, t]$, i.e. there is no action enabled on an urgent rendez-vous channel.
  - There is $i \in \{1, \dots, n\}$ such that there is $\bar{e} \in E_i$ and the ith automaton is in $\ell(\bar{e})$ and $\alpha(\bar{e}) = b!$, with $b \in B^u \cap B^b$, and $\nu + t' \not\models \varphi(\bar{e})$ for all $t' \in [0, t]$, i.e. there is no action enabled on an urgent broadcast channel.

- *Broadcast transition* $\langle \vec{\ell}, \nu \rangle \xrightarrow{\tau} \langle \vec{\ell'}, \nu' \rangle$ occurs if
  - there are a broadcast channel $b \in B^b$, an index $1 \leq i_0 \leq n$, an enabled edge $(\ell_{i_0}, b!, \varphi_{i_0}, \vec{r}_{i_0}, \ell'_{i_0}) \in E_{i_0}$, indices $1 \leq i_1, \dots, i_k \leq n$, $k \geq 0$, different from i_0, and edges (possibly enabled) $(\ell_{i_j}, b?, \varphi_{i_j}, \vec{r}_{i_j}, \ell'_{i_j}) \in E_{i_j}$, $i_j \in \{i_1, \dots, i_k\}$, such that
    - $\nu \models \varphi_{i_0} \wedge \varphi_{i_1} \wedge \dots \wedge \varphi_{i_k}$, i.e. the guards of the mentioned edges are satisfied.
    - $\vec{\ell'} = \vec{\ell}[\ell_{i_0} := \ell'_{i_0}][\ell_{i_1} := \ell'_{i_1}] \dots [\ell_{i_k} := \ell'_{i_k}]$.

- $\nu' = \nu[\vec{r}_{i_0}][\vec{r}_{i_1}]\ldots[\vec{r}_{i_k}], \nu' \models I_{i_0}(\ell'_{i_0}) \wedge I_{i_1}(\ell'_{i_1}) \wedge \cdots \wedge I_{i_k}(\ell'_{i_k})$.
- If there exists $\hat{k} \in \{1, \ldots, n\}$ such that $\ell_{\hat{k}} \in L^c_{\hat{k}}$, then there exists $\bar{k} \in \{i_0, i_1, \ldots, i_k\}$ such that $\ell_{\bar{k}} \in L^c_{\bar{k}}$.
- If for all $q \in \{1, \ldots, n\}$ such that $\hat{k} \neq q \neq \bar{k}$, there is $\bar{e} \in E_q$ such that the qth automaton is in $\ell(\bar{e})$ then $\rho(b) > \rho(chan(\alpha(\bar{e})))$, i.e. the broadcast transition induced by taking edges $e_{i_0}, e_{i_1}, \ldots e_{i_k}$ has a higher priority over any transition induced by taking edge $\bar{e}$.

- $\mathcal{C}_{ini} = \{\langle \vec{\ell}_{ini}, \nu_{ini} \rangle\} \cap Conf(\mathcal{N})$, where vector $\vec{\ell}_{ini}$ consists of the initial location of all component automata $\mathcal{A}_i$ and ν_{ini} assigns 0 to each clock $x \in \mathcal{X}_i$ and initial values to each variable $v \in V_i$, with $1 \leq i \leq n$. $\diamondsuit$

We introduce the following notation, which is not in [3]. For a configuration $s_k = \langle \vec{\ell}, \nu \rangle \in Conf(\mathcal{N})$, we write:

- $\vec{\ell}_{s_k}$, to denote the location vector $\vec{\ell}$ of s_k.
- ν_{s_k}, to denote the valuation ν of s_k.
- $\ell_{s_{k,i}}$, to denote the ith location in the vector $\vec{\ell}_{s_k}$.
- $s_k \models \varphi$, if and only if $\nu_{s_k} \models \varphi$, as already defined.

We denote by $s_0 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_n} s_n \in \mathcal{T}_e(\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n))$ a computation path of the transition system $\mathcal{T}_e(\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n))$, starting at $s_0 \in Conf(\mathcal{N})$. The set $Reach_{\mathcal{N}}$ contains all reachable configurations of network $\mathcal{N}$in other words $Reach_{\mathcal{N}}$ denotes the state-space of $\mathcal{N}$.

3.3 The Logic of Uppaal

Before introducing the logic of Uppaal we extend constraints over clocks and variables to include constraints over fresh logical variables. This extension is not in [3]. This extension will be used only in the logic of Uppaal, which we borrow from [3] (pages 176-180) as well and we extend it here.

Definition 17 *Constraints over Clocks, Integer and Logical Variables.*

Let $\mathcal{X}$ be a set of clocks. Let V be a set of integer variables disjoint from $\mathcal{X}$. Let $LogVars$ be a set of logical variables disjoint from $\mathcal{X}$ and V. The set $\Phi(\mathcal{X}, V, LogVars)$ of constraints over clocks, integer and logical variables with typical element φ is defined by the following syntax:

$$\varphi ::= \varphi_{clk} \mid \varphi_{int} \mid \varphi_1 \wedge \varphi_2.$$

Where $\varphi_{clk} \in \Phi(\mathcal{X})$ is a clock constraint, and $\varphi_{int} \in \Phi(V, LogVars)$ is an integer constraint using variables in V and $LogVars$. $\diamondsuit$

The following valuation of logical variables $\delta : \mathit{LogVars} \to \{0,1\}$ is a mapping that assigns to each logical variable $\theta \in \mathit{LogVars}$ a value in $\{0,1\}$. An assignment to θ is defined as follows:

$$\delta[\theta := d](z) = \begin{cases} d, & \text{if } z = \theta, \\ \delta(z), & \text{otherwise.} \end{cases}$$

Definition 18 *The Logic of Uppaal.*

The logic of Uppaal is a subset of the Timed Computation Tree Logic, tailored towards an efficient model checking procedure. Informally, this logic allow us to express that the following properties φ of configurations should hold along the computation paths of a given network

$$\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n) \tag{3.3.1}$$

of extended timed automata:

- $\exists \Diamond \varphi$ expresses that there exists a computation path along which eventually φ holds.
- $\forall \Box \varphi$ expresses that along all computation paths φ always holds.
- $\exists \Box \varphi$ expresses that there exists a computation path along which φ always holds.
- $\forall \Diamond \varphi$ expresses that along all computation paths φ eventually holds.
- $\varphi_1 \longrightarrow \varphi_2$ expresses that each occurrence of φ_1 eventually leads to an occurrence of φ_2.

Additionally, the above expressions can include existential quantification over logical variables $\theta_1, \ldots, \theta_n \in \mathit{LogVars}$, e.g. $\exists \Diamond \exists\, \theta_1, \ldots, \theta_n \bullet \varphi$. Formally the logic comprises *basic formulas BF, configuration formulas CF* (extended here to include existential quantification over logical variables in $\mathit{LogVars}$), and *path formulas PF*, divided into *existential path formulas EPF* and *universal path formulas APF*. The logic is defined by the following syntax:

$$\begin{array}{lll} \mathit{BF} & ::= & \mathcal{A}_i.\ell \mid \varphi, \\ \mathit{CF} & ::= & \mathit{BF} \mid \neg \mathit{CF} \mid \mathit{CF}_1 \wedge \mathit{CF}_2 \mid \exists\, \theta_1, \ldots, \theta_n \bullet \mathit{CF}, \\ \mathit{EPF} & ::= & \exists \Diamond \mathit{CF} \mid \exists \Box \mathit{CF}, \\ \mathit{APF} & ::= & \forall \Box \mathit{CF} \mid \forall \Diamond \mathit{CF} \mid \mathit{CF} \longrightarrow \mathit{CF}, \\ \mathit{PF} & ::= & \mathit{EPF} \mid \mathit{APF}. \end{array}$$

The basic formula $\mathcal{A}_i.\ell$ expresses that the automaton $\mathcal{A}_i$ of the network $\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ is at location ℓ, and basic formula φ is a constraint on clocks, integer and logical variables. In configuration formulas *CF* the logical connectives $\vee, \Longrightarrow$, and $\Longleftrightarrow$ are considered abbreviations. In path formulas *PF* the quantifiers $\exists$ and $\forall$ express existential and universal quantification over computation paths, respectively, and the modalities $\Diamond$ and $\Box$ express existential and universal quantification over configurations, respectively. For example the *PF*,

$$\exists \Diamond \mathcal{A}_i.\ell$$

expresses that there exists a computation path on which there exists a configuration where the automaton $\mathcal{A}_i$ is at location ℓ. In other words, the location ℓ is reachable in $\mathcal{A}_i$.

Given a path π of $\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ starting in the time-stamped configuration $\langle \vec{\ell_0}, \nu_0 \rangle, t_0$ of the form

$$\pi : \langle \vec{\ell_0}, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \vec{\ell_1}, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \langle \vec{\ell_2}, \nu_2 \rangle, t_2 \xrightarrow{\lambda_3} \ldots$$

and a value $t \in \mathit{Time}$ we denote by $\pi(t)$ the *set of configurations at time* t, defined as follows:

$$\pi(t) = \{\langle \vec{\ell}, \nu \rangle \mid \exists\, i \in \mathbb{N} \bullet (t_i \leq t \leq t_{i+1} \wedge \vec{\ell} = \vec{\ell_i} \wedge \nu = \nu_i + t - t_i)\}$$

Note that $\pi(t)$ is defined as a *set* because in π a sequence of transitions can occur at the same time. This set may be empty if the time stamps $t_0, t_1, t_2, t_3, \ldots$ do not form a real-time sequence, i.e. do not grow unboundedly. In that case there may be no index i with $t_i \leq t \leq t_{i+1}$.

The binary *satisfaction relation* $\models_\delta$, using the valuation of logical variables δ, between time stamped configurations of the network (3.3.1) and formulas F of the Uppaal logic, is written as $\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta F$ and defined inductively as follows:

$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta \mathcal{A}_i.\ell$	iff	$\ell_{0,i} = \ell$, i.e. the ith component of the location vector $\vec{\ell_0}$ is ℓ,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta \varphi$	iff	$\nu_0 \models_\delta \varphi$,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta \neg CF$	iff	$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \not\models_\delta CF$,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta CF_1 \wedge CF_2$	iff	$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta CF_1$ and $\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta CF_2$,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta$ $\exists\, \theta_1, \ldots, \theta_n \bullet CF$	iff	$\exists\, d_1, \ldots, d_n \in \{0, 1\} \bullet$ $\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_{\delta[\theta_1 := d_1, \ldots, \theta_n := d_n]} CF$,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta \exists \Diamond CF$	iff	$\exists$ path π of (3.3.1) starting in $\langle \vec{\ell_0}, \nu_0 \rangle, t_0$ $\exists\, \in \mathit{Time}, \langle \vec{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet t_0 \leq t$ $\wedge \langle \vec{\ell}, \nu \rangle \in \pi(t) \wedge \langle \vec{\ell}, \nu \rangle, t \models_\delta CF$,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta \forall \Box CF$	iff	$\forall$ path π of (3.3.1) starting in $\langle \vec{\ell_0}, \nu_0 \rangle, t_0$ $\forall\, t \in \mathit{Time}, \langle \vec{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet t_0 \leq t$ $\wedge \langle \vec{\ell}, \nu \rangle \in \pi(t) \Longrightarrow \langle \vec{\ell}, \nu \rangle, t \models_\delta CF$,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta \exists \Box CF$	iff	$\exists$ path π of (3.3.1) starting in $\langle \vec{\ell_0}, \nu_0 \rangle, t_0$ $\forall\, t \in \mathit{Time}, \langle \vec{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet t_0 \leq t$ $\wedge \langle \vec{\ell}, \nu \rangle \in \pi(t) \Longrightarrow \langle \vec{\ell}, \nu \rangle, t \models_\delta CF$,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta \forall \Diamond CF$	iff	$\forall$ path π of (3.3.1) starting in $\langle \vec{\ell_0}, \nu_0 \rangle, t_0$ $\exists\, t \in \mathit{Time}, \langle \vec{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet t_0 \leq t$ $\wedge \langle \vec{\ell}, \nu \rangle \in \pi(t) \wedge \langle \vec{\ell}, \nu \rangle, t \models_\delta CF$,
$\langle \vec{\ell_0}, \nu_0 \rangle, t_0 \models_\delta CF \longrightarrow CF$	iff	$\forall$ path π of (3.3.1) starting in $\langle \vec{\ell_0}, \nu_0 \rangle, t_0$ $\forall\, t \in \mathit{Time}, \langle \vec{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet t_0 \leq t$ $\wedge \langle \vec{\ell}, \nu \rangle \in \pi(t) \wedge \langle \vec{\ell}, \nu \rangle, t \models_\delta CF$ implies $\langle \vec{\ell}, \nu \rangle, t \models_\delta \forall \Diamond CF$.

We lift the satisfaction relation $\models_\delta$ to networks $\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$, existential path formulas *EPF*, and universal path formulas *APF* as follows:

$$\begin{array}{lll} \mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n) \models_\delta \mathit{EPF} & \text{iff} & \langle \vec{\ell_0}, \nu_0 \rangle, 0 \models_\delta \mathit{EPF} \text{ for some } \langle \vec{\ell_0}, \nu_0 \rangle \in \mathcal{C}_{ini}, \\ \mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n) \models_\delta \mathit{APF} & \text{iff} & \langle \vec{\ell_0}, \nu_0 \rangle, 0 \models_\delta \mathit{APF} \text{ for all } \langle \vec{\ell_0}, \nu_0 \rangle \in \mathcal{C}_{ini}, \end{array}$$

where $\mathcal{C}_{ini}$ is the set of initial configurations in $\mathcal{T}_e(\mathcal{N}(\mathcal{A}_1, \ldots, \mathcal{A}_n))$, the transition system of the network. Recall that $\mathcal{C}_{ini}$ contains at most one element. If $\mathcal{C}_{ini} = \emptyset$ the formula *EPF* is never satisfied whereas *APF* is trivially satisfied. We may omit δ from $\models_\delta$ if it is clear from the context. ♢

Chapter 4

Quasi-Equal Clocks

4.1 Introduction

In this chapter we recall the formal notion of quasi-equal clocks and the lemma that shows that *quasi-equality* is an equivalence relation.

4.2 Quasi-Equal Clocks

Real-timed clocks and the state-space they induce, present an obstacle for model checking industrial-size networks of timed automata. We propose to reduce the model checking effort in networks where we apply our quasi-equal clocks reduction approach. To this end we present the notion of quasi-equal clocks, and later an algorithm to reduce the number of quasi-equal clocks.

Intuitively, two clocks are quasi-equal if and only if their values differ only at time points in which those clocks are reset. We present the following two examples where quasi-equal clocks occur.

Example 1 (Network with Quasi-Equal Clocks) *Consider the network $\mathcal{N}$ in Figure 4.1. Network $\mathcal{N}$ consists of automata $\mathcal{A}_1$ and $\mathcal{A}_2$ with respective clocks x and y. After delaying ten time units at their respective initial locations ℓ_0 and ℓ_2, automata $\mathcal{A}_1$ and $\mathcal{A}_2$ interleave by taking their unique edges which reset their respective clocks. This interleaving induces configurations where clocks x and y differ on their values. However, after being reset both clocks evolve with the same value. Since the values of clocks x and y only differ at the time point when they are reset, therefore they are quasi-equal clocks.* ♢

FIGURE 4.1: Network $\mathcal{N}$ with quasi-equal clocks x and y.

The following is an example of a single automaton with quasi-equal clocks. Since our quasi-equal clocks reduction approach is independent from the structure of the timed automata where quasi-equal clocks occur, we transform as well single automaton with quasi-equal clocks.

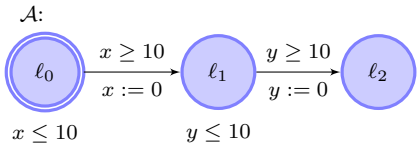


FIGURE 4.2: Automaton $\mathcal{A}$ with quasi-equal clocks x and y.

Example 2 (Single Automaton with Quasi-Equal Clocks) *Consider the single automaton $\mathcal{A}$ in Figure 4.2. Automaton $\mathcal{A}$ has clocks x and y. After delaying ten time units at the initial location ℓ_0, automaton $\mathcal{A}$ resets the clock x and transits to location ℓ_1. Note that at ℓ_1 the value of the clock y is ten, thus, no delay greater than zero time units is allowed at that location, thus, $\mathcal{A}$ transits to location ℓ_2 and resets the clock y. In $\mathcal{A}$ there is no interleaving of transitions. However, there is a configuration where the clocks x and y differ on their values. Since the values of clocks x and y only differ at the time point when they are reset, therefore they are quasi-equal clocks.* ♢

We recall from [4] the following definition of quasi-equal clocks. Note that this definition applies as well for quasi-equal clocks in single automata.

Definition 19 *Quasi-Equal Clocks.*

Let $\mathcal{N}$ be a network of timed automata with a set of clocks $\mathcal{X}$. Two clocks $x, y \in \mathcal{X}$ are called *quasi-equal*, denoted by $x \simeq y$, if and only if for all computation paths of $\mathcal{N}$, the values of x and y are equal, or the value of one of both is equal to 0, i.e. if

$$\forall \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \ldots \in Paths(\mathcal{N}) \forall\, i \in \mathbb{N} \bullet$$

$$\nu_i \models (x = 0 \vee y = 0 \vee x = y).$$

♢

In the following lemma we show that quasi-equality is an equivalence relation by which we obtain equivalence classes of quasi-equal clocks. By replacing all quasi-equal clocks of one equivalence class by a single representative clock, we obtain a transformed network which models the intended behaviour of the original network.

Lemma 1 *Quasi-Equality is an Equivalence Relation.*

Let $\mathcal{N}$ be a network of timed automata, and $\mathcal{X}$ its set of clocks. The quasi-equality relation $\simeq \subseteq \mathcal{X} \times \mathcal{X}$ is an equivalence relation.

Proof. We show that $\simeq \subseteq \mathcal{X} \times \mathcal{X}$ is an equivalence relation by showing that $\simeq$ is reflexive, symmetric and transitive.

1. $\simeq$ *is reflexive.*
   Let $x \in \mathcal{X}$ be a clock. We show that $x \simeq x$. By Definition 19,

$$\forall \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \ldots \in Paths(\mathcal{N}) \forall\, i \in \mathbb{N} \bullet$$

$$\nu_i \models (x = 0 \vee x = 0 \vee x = x).$$

Then by the reflexivity property of " $=$ ", $x = x$ always holds. Thus $x \simeq x$. Hence $\simeq$ is *reflexive*.

2. $\simeq$ *is symmetric*.
Let $x, y \in \mathcal{X}$ be clocks. We show that $x \simeq y \implies y \simeq x$. Assume $x \simeq y$. Then

$$\forall \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \ldots \in Paths(\mathcal{N}) \forall\, i \in \mathbb{N} \bullet$$
$$\nu_i \models (x = 0 \vee y = 0 \vee x = y).$$

By the symmetry property of " $=$ " and the associativity property of " $\vee$ ",

$$(x = 0 \vee y = 0 \vee x = y) \iff (y = 0 \vee x = 0 \vee y = x).$$

Thus

$$\forall \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \ldots \in Paths(\mathcal{N}) \forall\, i \in \mathbb{N} \bullet$$
$$\nu_i \models (y = 0 \vee x = 0 \vee y = x).$$

Thus $y \simeq x$. Hence $\simeq$ is *symmetric*.

3. $\simeq$ *is transitive*.
Let $x, y, z \in \mathcal{X}$ be clocks. We show that $x \simeq y \wedge y \simeq z \implies x \simeq z$. Assume $x \simeq y \wedge y \simeq z$. Then

$$\forall \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \ldots \in Paths(\mathcal{N}) \forall\, i \in \mathbb{N} \bullet$$
$$\nu_i \models (x = 0 \vee y = 0 \vee x = y) \wedge (y = 0 \vee z = 0 \vee y = z).$$

To show that $x \simeq z$, we prove by induction on the ith element $\pi_i = (\langle \ell_i, \nu_i \rangle, t_i)$ of a path π, that if the valuation ν_i satisfies the following strong constraint defined in terms of x, y and z,

$$\phi := (x = 0 \vee y = 0 \vee x = y) \wedge (y = 0 \vee z = 0 \vee y = z) \wedge$$
$$(x = 0 \vee z = 0 \vee x = z),$$

implies that the valuation ν_{i+1} satisfies ϕ. In other words, we show that

$$\forall \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \ldots \in Paths(\mathcal{N}) \forall\, i \in \mathbb{N} \bullet$$
$$\nu_{i+1} \models \phi.$$

Base Case π_0*:* In $\pi_0 = (\langle \ell_0, \nu_0 \rangle, 0)$ the values of the clocks x, y and z are $\nu_0(x) = 0$, $\nu_0(y) = 0$ and $\nu_0(z) = 0$, respectively. Thus

$$\nu_0 \models \phi.$$

Induction Step $\pi_i \to \pi_{i+1}$*:* Assume in $\pi_i = (\langle \ell_i, \nu_i \rangle, t_i)$ that

$$\nu_i \models \phi.$$

We prove in $\pi_{i+1} = (\langle \ell_{i+1}, \nu_{i+1} \rangle, t_{i+1})$ that the following holds

$$\nu_{i+1} \models \phi.$$

We distinguish two cases based on the values of the clocks x, y, z at π_i.

Case 1: at π_i clocks x, y and z have the same value. We distinguish the following changes of the values of the clocks x, y, z, after the following transitions:

After the transition $(\langle \ell_i, \nu_i \rangle, t_i) \xrightarrow{\tau} (\langle \ell_{i+1}, \nu_{i+1} \rangle, t_{i+1})$, the value of one clock from x, y, z is different than the values from the other two. Then,

$$\nu_{i+1} \models \phi.$$

After the transition $(\langle \ell_i, \nu_i \rangle, t_i) \xrightarrow{\tau} (\langle \ell_{i+1}, \nu_{i+1} \rangle, t_{i+1})$, clocks x, y, z have the same value. Then,

$$\nu_{i+1} \models \phi.$$

After the transition $(\langle \ell_i, \nu_i \rangle, t_i) \xrightarrow{t} (\langle \ell_{i+1}, \nu_{i+1} + t \rangle, t_i + t)$, the clocks x, y, z have the same value. Then,

$$\nu_{i+1} \models \phi.$$

Case 2: at $\pi_i, 0$ is the value of one clock from x, y, z, and the other two clocks have the same value different than 0. Or, 0 is the value of two clocks from x, y, z and the value of the third clock is different than 0.

After the transition $(\langle \ell_i, \nu_i \rangle, t_i) \xrightarrow{\tau} (\langle \ell_{i+1}, \nu_{i+1} \rangle, t_{i+1})$, the value of one clock from x, y, z is different than the value from the other two. Then,

$$\nu_{i+1} \models \phi.$$

After the transition $(\langle \ell_i, \nu_i \rangle, t_i) \xrightarrow{\tau} (\langle \ell_{i+1}, \nu_{i+1} \rangle, t_{i+1})$, clocks x, y, z have the same value. Then,

$$\nu_{i+1} \models \phi.$$

After the transition $(\langle \ell_i, \nu_i \rangle, t_i) \xrightarrow{t} (\langle \ell_{i+1}, \nu_{i+1} + t \rangle, t_i + t)$,

$$\nu_{i+1} \not\models \phi.$$

That is, if, for instance, a *delay transition* of $t = 10$ occurs then the clocks x, y, z could have the values $\nu_{i+1}(x) = 10$, $\nu_{i+1}(y) = 20$, $\nu_{i+1}(z) = 20$, which does not satisfy ϕ, and violates the assumption that $x \simeq y \wedge y \simeq z$, which means that the following does not hold,

$$\forall \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \ldots \in Paths(\mathcal{N}) \forall\, i \in \mathbb{N} \bullet$$
$$\nu_i \models (x = 0 \vee y = 0 \vee x = y) \wedge (y = 0 \vee z = 0 \vee y = z).$$

Hence, a delay transition when the clocks x, y, z have the values at π_i as described in Case 2, is a contradiction to the assumption that $x \simeq y \wedge y \simeq z$.

Since we showed that

$$\forall \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \ldots \in Paths(\mathcal{N}) \forall\, i \in \mathbb{N} \bullet \nu_{i+1} \models \phi.$$

Thus $x \simeq z$. Thus $\simeq$ is *transitive*. □

For a given network we obtain the set of equivalence classes of quasi-equal clocks by using $\simeq$ over the set of clocks of that network. From each equivalence class we choose a representative clock and use it in a transformed network to model the intended behaviour of the original network.

Definition 20 *The Set $\mathcal{EC}_{\mathcal{N}}$.*

Let $\mathcal{N}$ be a network of timed automata and $\mathcal{X}$ its set of clocks. The set $\mathcal{EC}_{\mathcal{N}}$ is the set of equivalence classes of quasi-equal clocks of $\mathcal{N}$, or the quotient set of $\mathcal{X}$ by $\simeq$, i.e. if

$$\mathcal{EC}_{\mathcal{N}} = \{Y \in \mathcal{X}/\simeq\}.$$

We choose a representative clock of each $Y \in \mathcal{EC}_{\mathcal{N}}$ denoted by $rep(Y)$, or $rep(x)$ where $x \in Y$. $\diamondsuit$

Chapter 5

Transformation of Automata with Quasi-Equal Clocks.

5.1 Introduction

In this chapter we present two distinctions for edges resetting quasi-equal clocks, namely, *simple* and *complex edges*. These distinctions allow us to provide qualitatively different savings wrt. space and time.

We present three transformations as well. The first is a transformation function for extended clock constraints in guards and invariants using quasi-equal clocks which we use as a helper function in the second transformation. The second is a transformation algorithm which we use to reduce clocks in any timed automaton or network with quasi-equal clocks. Finally, the third is a transformation function which we use for properties over an original automaton or network with quasi-equal clocks.

5.2 Simple and Complex Resetting Edges

In the following we make a distinction of edges resetting quasi-equal clocks. Based on this distinction we are able to propose transformations for those edges and thereby obtain in transformed networks qualitatively different savings wrt. space and time.

We distinguish edges which reset quasi-equal clocks in two groups, namely, simple and complex (resetting) edges. Simple edges are characterised by resetting exclusively quasi-equal clocks, and enforcing delays greater than 0 time units at their origin and destination locations. These edges are independent, i.e. they do not communicate through channels or shared variables, therefore, they are able to interleave in all possible orders at a given time point. Due to their independence and since simple edges do not update information to keep track of, i.e. values of data variables, we are able to synchronise them by using a broadcast transition. This broadcast transition saves all configurations induced by each single transition justified by a simple edge. Complex edges, as opposed to simple ones, either update data variables or are not independent or do not enforce delays greater than 0 time units at their origin or destination locations. Therefore, in general we are not able to synchronise complex edges. We preserve in transformed networks all configurations induced by transformed complex edges, otherwise, transformed networks do not reflect the truth-value of all queries on original networks.

Definition 21 *Pre/Post Delayed Edge.*

Let $\mathcal{N}$ be a network of timed automata with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. An edge $e = (\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{N})$ is called *pre/post-delayed* if and only if time must pass in ℓ and ℓ' before and after e is taken, respectively, i.e. if

$$Paths(\mathcal{N}) = Paths(\Upsilon(\mathcal{N})),$$

where Υ is a transformation that adds a fresh clock x in $\mathcal{N}$, and for each edge incoming to and outgoing from ℓ, a reset $x := 0$, and to the guard φ the condition $x > 0$, and we add $x > 0$ as well to guards of all outgoing edges of ℓ'. Note that the added clock constraints enforce delays greater than zero time units at locations ℓ and ℓ'. We use $DlayEdges_{\mathcal{N}}$ to denote the set of pre/post-delayed edges of $\mathcal{N}$. $\diamondsuit$

Definition 22 *Simple and Complex (Resetting) Edges.*

Let $\mathcal{N}$ be a network of timed automata with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{A} \in \mathcal{N}$ be an automaton. Let $e = (\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{A})$ be an edge which resets at least one quasi-equal clock, i.e. $clocks(\vec{r}) \cap Y \neq \emptyset$ for some $Y \in \mathcal{EC}_{\mathcal{N}}$. Edge e is called *simple edge* if and only if

1. it is of the form $(\ell, \tau, x \geq c, \langle x := 0 \rangle, \ell')$ for some local clock $x \in \mathcal{X}(\mathcal{A})$,
2. the invariant of ℓ is $x \leq c$,
3. it is pre/post-delayed, i.e. $e \in DlayEdges_{\mathcal{N}}$,
4. it is the only outgoing edge of ℓ, i.e. $\forall\, e_1 = (\ell_1, \alpha_1, \varphi_1, \vec{r}_1, \ell_1') \in E(\mathcal{A}) \bullet \ell_1 = \ell \implies e = e_1$, and it is the only incoming edge into ℓ', i.e. $\forall\, e_1 = (\ell_1, \alpha_1, \varphi_1, \vec{r}_1, \ell_1') \in E(\mathcal{A}) \bullet \ell_1' = \ell' \implies e = e_1$.

Otherwise, e is called *complex edge*. We use $SimpEdges_Y(\mathcal{A})$ to denote the set of simple edges of $\mathcal{A}$ using a clock from Y. $CompEdges_Y(\mathcal{A})$ is used to denote the set of those complex edges which reset at least one clock from Y. We use $SimCompEdges_Y(\mathcal{A}) = SimpEdges_Y(\mathcal{A}) \cup CompEdges_Y(\mathcal{A})$ to denote the set of resetting edges of $\mathcal{A}$ which reset clocks from Y. $\diamondsuit$

5.3 A Running Example

Example 3 (Network $\mathcal{N}_1$) *Consider the network $\mathcal{N}_1$ in Figure 5.1. Network $\mathcal{N}_1$ consists of automata $\mathcal{A}_1$ and $\mathcal{A}_2$ with respective clocks x and y, rendez-vous channel c, and global variable a. After delaying ten time units at their respective initial locations, automata $\mathcal{A}_1$ and $\mathcal{A}_2$ interleave by taking their simple edges which exclusively reset their respective clocks. This interleaving induces configurations where clocks x and y differ on their values. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$ after a delay of five time units at locations ℓ_1 and ℓ_5 interleave once again by taking their complex edges which reset their respective clocks together with updates of the variable a. Note that automata $\mathcal{A}_1$ and $\mathcal{A}_2$ can reset once again their respective clocks and transit simultaneously to their respective locations ℓ_3 and ℓ_7 at any time, even without delaying at locations ℓ_2 and ℓ_6. Since the valuations of clocks x and y only differ at the time point when they are reset, therefore they are quasi-equal clocks.* $\diamondsuit$

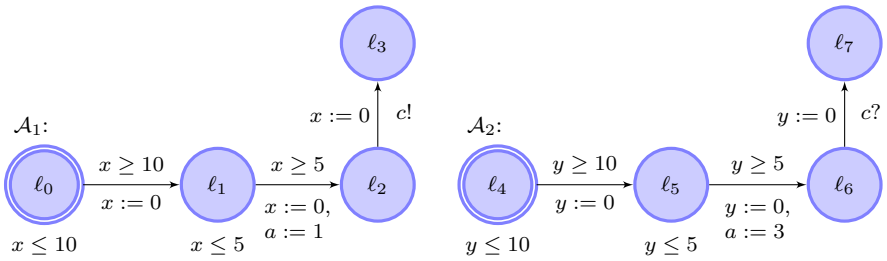


FIGURE 5.1: Network $\mathcal{N}_1$ with quasi-equal clocks x and y.

Our approach in this work transforms any single automaton and network with quasi-equal clocks. In this work we have removed any semantical and syntactical assumption on networks in order to soundly apply the reduction of clocks, as opposed to the approaches in [4, 7, 8] where we impose strong semantical and syntactical assumptions on networks. For instance, $\mathcal{N}_1$ cannot be transformed by the approach from [7], since by that approach: (a) the outgoing edges of locations ℓ_2 and ℓ_6 do not fulfill the syntactical pattern of an edge resetting quasi-equal clocks, i.e. there are no clock constraints that guard those edges, and the origin locations of those edges have no invariants; and (b) there must be a delay greater than zero time units at the origin location of any edge resetting a quasi-equal clock.

In the following we distinguish configurations of networks with quasi-equal clocks, namely, *stable* and *unstable configurations* per equivalence class Y. Intuitively, stable configurations are configurations where all clocks from Y have the same value, thus, in particular the same value as the representative $rep(Y)$. Unstable configurations are configurations where some clocks from Y have been reset and some not yet, so each clock from Y either has the value 0 or the same value as $rep(Y)$.

We make this distinction of configurations since it help us, for instance, to analyse which configurations are preserved in transformed networks after reducing quasi-equal clocks. Then, based on that analysis later we define a transformation for querying those preserved configurations.

Definition 23 *Stable Configuration.*

Let $\mathcal{N}$ be a network of timed automata with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. A configuration $s \in Conf(\mathcal{N})$ is called *stable* wrt. $Y \in \mathcal{EC}_{\mathcal{N}}$ if and only if all clocks in Y have the same value in s, i.e. if

$$\forall\, x \in Y \bullet \nu_s(x) = \nu_s(rep(x)).$$

We use $StableConf_Y$ to denote the set of all configurations that are stable wrt. Y. A configuration not in $StableConf_Y$ is called *unstable* wrt. Y. $\diamondsuit$

5.4 Reducing Quasi-Equal Clocks in Timed Automata

We now present the transformation for extended clock constraints in guards and invariants using quasi-equal clocks. Recall that we distinguish stable and unstable configurations per equivalence class Y. We use a fresh boolean token t_x for each quasi-equal clock $x \in Y$ to encode clock values in unstable configurations. Configurations in a transformed network $\mathcal{N}'$ where token t_x is 1, encode configurations of the respective original network $\mathcal{N}$ where $x = rep(x)$ holds, while the token being 0 encodes that x has already been reset at the current time point and thus has value 0. Function Γ (cf. Definition 24) transforms guards and invariants based on this encoding.

Definition 24 (Function Γ)
Let $\mathcal{N}$ be a network. Let $Y, W \in \mathcal{EC}_{\mathcal{N}}$ be sets of quasi-equal clocks of $\mathcal{N}$, $x \in Y$ and $y \in W$ clocks. Let $t_x, t_y \notin V(\mathcal{N})$ be boolean variables. Given a clock constraint φ_{clk}, we define:

$$\Gamma_0(\varphi_{clk}) := \begin{cases} ((rep(x) \sim c \ \wedge \ t_x) \vee (0 \sim c \ \wedge \ \neg t_x)) & \text{, if } \varphi_{clk} = x \sim c, \\ \big((rep(x) - rep(y) \sim c \ \wedge \ t_x \ \wedge \ t_y) & \text{, if } \varphi_{clk} = x - y \sim c, \\ \quad \vee \ (0 - rep(y) \sim c \ \wedge \ \neg t_x \ \wedge \ t_y) & \\ \quad \vee \ (rep(x) - 0 \sim c \ \wedge \ t_x \ \wedge \ \neg t_y) & \\ \quad \vee \ (0 \sim c \ \wedge \ \neg t_x \ \wedge \ \neg t_y)\big) & \\ \Gamma_0(\varphi_1) \wedge \Gamma_0(\varphi_2) & \text{, if } \varphi_{clk} = \varphi_1 \wedge \varphi_2. \end{cases}$$

We get the transformation Γ by setting $\Gamma(\varphi_{clk} \wedge \psi_{int}) := \Gamma_0(\varphi_{clk}) \wedge \psi_{int}$. $\diamondsuit$

Before we present the transformation algorithm we use to reduce quasi-equal clocks in timed automata, we introduce the source and destination locations, so-called *reset locations*, of simple and complex edges of automata. Our algorithm mainly operates on those locations.

Definition 25 *Reset and Reset Successor Locations.*
Let $\mathcal{N}$ be a network of timed automata with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. Let $e = (\ell, \alpha, \varphi, \vec{r}, \ell') \in SimCompEdges_Y(\mathcal{N})$ be a reset edge for some $Y \in \mathcal{EC}_{\mathcal{N}}$. Then location ℓ (ℓ') is called *reset (successor) location* wrt. Y. We use $RstLocs^-_Y(\mathcal{N})$ ($RstLocs^+_Y(\mathcal{N})$) to denote the set of reset (successor) locations wrt. Y. We set $RstLocs^-(\mathcal{N}) := \bigcup_{Y \in \mathcal{EC}_{\mathcal{N}}} RstLocs^-_Y(\mathcal{N})$ and similarly $RstLocs^+(\mathcal{N})$. $\diamondsuit$

Definition 26 *Reset Automata.*
Let $\mathcal{N}$ be a network of timed automata with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. We use $ResetAut_Y(\mathcal{N})$ to denote the set of automata in $\mathcal{N}$ which have simple or complex resetting edges wrt. $Y \in \mathcal{EC}_{\mathcal{N}}$, i.e. if $ResetAut_Y(\mathcal{N}) = \{\mathcal{A} \in \mathcal{N} \mid SimCompEdges_Y(\mathcal{A}) \neq \emptyset\}$.

We use $ResetAut(\mathcal{N}) = \bigcup_{Y \in \mathcal{EC}_{\mathcal{N}}} ResetAut_Y(\mathcal{N})$ to denote the set of all automata in $\mathcal{N}$ which have simple or complex resetting edges. $\diamondsuit$

Our transformation algorithm takes two inputs, a network $\mathcal{N}$ and a list of all equivalence classes of quasi-equal clocks of $\mathcal{N}$, $\mathcal{EC}^{prio}_{\mathcal{N}}$, where those equivalence classes are elements of $\mathcal{EC}_{\mathcal{N}}$. Our algorithm outputs a transformed network $\mathcal{N}'$ which from each $Y \in \mathcal{EC}_{\mathcal{N}}$ uses only the representative clock $rep(Y)$. Network $\mathcal{N}'$ reflects the truth-value of all queries on $\mathcal{N}$.

Definition 27 (Transformation Algorithm $\mathcal{K}$)
Let $\mathcal{N} = \{\mathcal{A}_1, \ldots, \mathcal{A}_q\}$ be a network with a set of equivalence classes of quasi-equal clocks, $\mathcal{EC}_\mathcal{N}$. Let $\mathcal{EC}_\mathcal{N}^{prio}$ be a list of all elements of $\mathcal{EC}_\mathcal{N}$. The output of $\mathcal{K}$ is $\mathcal{N}' = \{\mathcal{K}(\mathcal{A}_1, \mathcal{EC}_\mathcal{N}^{prio}), \ldots, \mathcal{K}(\mathcal{A}_q, \mathcal{EC}_\mathcal{N}^{prio})\} \cup \{\mathcal{R}_Y \mid Y \in \mathcal{EC}_\mathcal{N}\}$, where $\mathcal{K}(\mathcal{A}, \mathcal{EC}_\mathcal{N}^{prio}) = (L(\mathcal{A}), B', \mathcal{X}', V', I', E_{cn} \cup E_s, \ell_{ini})$ such that:

- $B' = B(\mathcal{A}) \cup \{reset_Y, return, u_Y \mid \mathcal{A} \in ResetAut_Y(\mathcal{N}), Y \in \mathcal{EC}_\mathcal{N}\}$, i.e. the fresh broadcast channels $reset_Y$ and $return$, and the fresh urgent broadcast channel u_Y are added for each equivalence class whose clocks are updated by $\mathcal{A}$. Priorities on channels u_Y, $return$ and $reset_Y$ are assigned as follows. For all $Y \in \mathcal{EC}_\mathcal{N}$, channels $reset_Y$ and u_Y from the same Y have the same priority. Among equivalence classes from $\mathcal{EC}_\mathcal{N}$, the priorities for those two kinds of channels follow the order given in $\mathcal{EC}_\mathcal{N}^{prio}$ by the user of Algorithm $\mathcal{K}$. All other channels including $return$ as well as local transitions, have the same priority which is smaller than the priority assigned to channels $reset_Y$ and u_Y.
- $\mathcal{X}' = \{rep(x) \mid x \in \mathcal{X}(\mathcal{A})\}$, i.e. only representative clocks are used.
- $V' = V(\mathcal{A}) \cup \{t_x \mid x \in Y, Y \in \mathcal{EC}_\mathcal{N}, \mathcal{A} \in ResetAut_Y(\mathcal{N})\} \cup \{s_Y^\mathcal{A} \mid SimpEdges_Y(\mathcal{A}) \neq \emptyset, Y \in \mathcal{EC}_\mathcal{N}\}$, i.e. one fresh boolean token t_x for each quasi-equal clock x is added (initial value is one), and a fresh boolean simple-edge indicator $s_Y^\mathcal{A}$ (initial value is one if and only if the initial location of $\mathcal{A}$ is a reset location of a simple edge).
- $I' = \{\ell \mapsto \Gamma(I(\ell)) \mid \ell \in L(\mathcal{A})\}$, i.e. invariants are transformed with Γ to consider representative clocks and tokens related to quasi-equal clocks.
- Complex and non-resetting edges are transformed as follows, and the resulting edges contained in E_{cn}. Guards are transformed using Γ. Reset vectors are transformed to consider tokens instead of the original clock, and extended by r_1 as book-keeping for the simple-edge indicator, where $r_1(\ell')$ yields the update $s_Y^\mathcal{A} := 1$ if ℓ' is the origin location of a simple resetting edge, and ϵ otherwise.

$$E_{cn} = \{(\ell, \alpha, \Gamma(\varphi), \vec{r}[x := 0/t_x := 0 \mid x \in Y, Y \in \mathcal{EC}_\mathcal{N}]; r_1(\ell'), \ell') \mid (\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{A}) \setminus SimpEdges_Y(\mathcal{A}), Y \in \mathcal{EC}_\mathcal{N}\}.$$

- Simple edges are transformed as follows (cf. Figure 5.2 (a) and (b)), and the resulting edges contained in E_s. We place in the transformed simple edge an output on $reset_Y$, we transform its guard by using Γ, and its reset vector is transformed similarly as in transformed complex edges. Additionally, we copy the transformed simple edge and instead of an output on $reset_Y$, we place an input on $reset_Y$.

$$E_s = \{(\ell, reset_Y!, \Gamma(x \geq c), t_x := 0; r_1(\ell'), \ell'), (\ell, reset_Y?, \Gamma(x \geq c), t_x := 0; r_1(\ell'), \ell') \mid (\ell, \tau, x \geq c, \langle x := 0 \rangle, \ell') \in SimpEdges_Y(\mathcal{A}), Y \in \mathcal{EC}_\mathcal{N}\}.$$

For each $Y \in \mathcal{EC}_{\mathcal{N}}$, the resetter $\mathcal{R}_Y = (L, B, \mathcal{X}, V, I, E, \ell_{ini\mathcal{R}_Y})$ is created as follows (cf. Figure 5.2.(c)),

- $L = \{\ell_{ini\mathcal{R}_Y}, \ell_{nst\mathcal{R}_Y}, \mathit{Tlock}_Y\}$.
- $B = \{\mathit{reset}_Y, \mathit{return}, u_Y\}$, i.e. the broadcast channels reset_Y and return, the urgent broadcast channel u_Y are added. Priorities on channels u_Y, return and reset_Y are similarly assigned as above explained.
- $\mathcal{X} = \{\mathit{rep}(Y), z\}$, i.e. the set of clocks consists of the representative clock $\mathit{rep}(Y)$, and the fresh clock z exclusively used in invariants of urgent and committed locations.
- $V = \{t_x, \mathit{prio}_Y \mid x \in Y\} \cup \{s_Y^{\mathcal{A}} \mid \mathit{SimpEdges}_Y(\mathcal{A}) \neq \emptyset, \mathcal{A} \in \mathcal{N}\}$, i.e. the set of variables contains one boolean token t_x for each quasi-equal clock x, together with one fresh boolean priority variable prio_Y (initial value is zero), and a boolean simple-edge indicator $s_Y^{\mathcal{A}}$ for each automaton with simple edges.
- $I = \{\ell_{ini\mathcal{R}_Y} \mapsto \mathit{true}\} \cup \{\ell \mapsto z \leq 0 \mid \ell \in \{\ell_{nst\mathcal{R}_Y}, \mathit{Tlock}_Y\}\}$, i.e. the invariant of location $\ell_{ini\mathcal{R}_Y}$ is true, and the invariants of locations $\ell_{nst\mathcal{R}_Y}$ and Tlock_Y use the fresh clock z. Note that this clock is not added by this algorithm but internally used by Uppaal in urgent locations.
- $$
\begin{aligned}
E = \{ & \\
& (\ell_{ini\mathcal{R}_Y}, u_Y!, (\sum_{x \in Y} t_x = 0), \langle \mathit{prio}_Y := 1, z := 0\rangle, \ell_{nst\mathcal{R}_Y}), \\
& (\ell_{ini\mathcal{R}_Y}, \mathit{reset}_Y?, \mathit{true}, \langle \mathit{prio}_Y := 1, z := 0\rangle, \ell_{nst\mathcal{R}_Y}), \\
& (\ell_{nst\mathcal{R}_Y}, \tau, \mathit{go}(\mathcal{EC}_{\mathcal{N}}), \langle \mathit{prio}_Y := 0, z := 0\rangle, \mathit{Tlock}_Y), \\
& (\ell_{nst\mathcal{R}_Y}, \mathit{return}!, (\mathit{blk}(\mathcal{EC}_{\mathcal{N}}) \wedge \mathit{prties}(\mathcal{EC}_{\mathcal{N}})), \langle\rangle \\
& \quad [\mathit{rep}(Y) := 0, \mathit{prio}_Y := 0, t_{x_1} := 1, \ldots, t_{x_n} := 1 \mid x_1, \ldots, x_n \in Y], \\
& \quad \ell_{ini\mathcal{R}_Y}) \\
& (\ell_{nst\mathcal{R}_Y}, \mathit{return}?, \mathit{blk}(\mathcal{EC}_{\mathcal{N}}), \langle\rangle[\mathit{rep}(Y) := 0, \\
& \quad \mathit{prio}_Y := 0, t_{x_1} := 1, \ldots, t_{x_n} := 1 \mid x_1, \ldots, x_n \in Y], \ell_{ini\mathcal{R}_Y}) \\
& \quad \},
\end{aligned}
$$

  where $\mathit{blk}(\mathcal{EC}_{\mathcal{N}}) := \sum_{x \in Y} t_x = 0 \wedge \bigwedge_{W \in \mathcal{EC}_{\mathcal{N}} \setminus \{Y\}} (\sum_{w \in W} t_w = 0 \vee \sum_{w \in W} t_w = |W|)$ is a function used to allow transitions if and only if all equivalence classes in $\mathcal{EC}_{\mathcal{N}}$ are stable. Given $\mathcal{EC}_{\mathcal{N}}^{prio} = \langle Y_1, \ldots, Y_m\rangle$, for some index $1 \leq j \leq m$, such that $Y_j = Y$, then $\mathit{prties}(\mathcal{EC}_{\mathcal{N}}) := \neg \mathit{prio}_{Y_{j+1}} \wedge \cdots \wedge \neg \mathit{prio}_{Y_m}$, is a function implementing a mechanism to prioritise transitions which is based on the ordering given for equivalence classes in $\mathcal{EC}_{\mathcal{N}}^{prio}$.

  Function $\mathit{go}(\mathcal{EC}_{\mathcal{N}}) := \mathit{prties}(\mathcal{EC}_{\mathcal{N}}) \wedge \bigvee_{x \in Y} \mathit{sum}(x)$ uses the output of $\mathit{prties}(\mathcal{EC}_{\mathcal{N}})$, and the output of function $\mathit{sum}(x)$, i.e. false if $\neg\exists\, 1 \leq i \leq q\ \exists\, (\ell, \alpha, \varphi, \vec{r}, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_i) \bullet x \in \mathit{clocks}(\vec{r})$, or $(s_Y^{\mathcal{A}_i} \wedge t_x)$, otherwise. Function $\mathit{sum}(x)$ is used to detect from values of the boolean variables $s_Y^{\mathcal{A}_i}$ and t_x, cases where time is stopped (and no delay transitions greater than 0 are possible) either at origin or at destination locations of simple edges. $\diamondsuit$

From Definition 27 we note the following. As presented in [4, 7, 8, 9], transformed networks include a resetter automaton $\mathcal{R}_Y$ which we use to reset of the clock $rep(Y)$. Resetter $\mathcal{R}_Y$ has the location $\ell_{nst\mathcal{R}_Y}$ which, as in [7, 8, 9], is used to encode configurations induced by all possible permutations of resets of clocks in Y. For instance, assuming $\ell_{nst\mathcal{R}_Y}$ in $\mathcal{N}'$ represents both cases for a simple edge, that it has already been taken or not, and that the clock $x \in Y$ reset by this edge is already 0 or $x = rep(Y)$.

Similar to [8, 9] transformed automata use synchronisations to indicate $\mathcal{R}_Y$ when to reset the clock $rep(Y)$ as opposed to encoding time points in $\mathcal{R}_Y$ for resetting $rep(Y)$ as in [7]. Moreover, in order to reflect in $\mathcal{N}'$ the truth-value of all queries on $\mathcal{N}$, and given that simple edges are independent, i.e. communication through channels or shared variables is not allowed since it could enforce a certain dependency among those edges, we prioritise transformed simple edges over all other edges as follows. At a given time point where the clock $rep(Y)$ is reset, we allow enabled transformed simple edges wrt. Y (if any) to be taken before any other edge that in $\mathcal{N}$ could be enabled at that time point. This prioritisation is implemented in two mechanisms executed by $\mathcal{R}_Y$ which together with an extra mechanism to reduce interleavings among resetters, are described as follows (cf. Figure 5.2).

1. *Broadcast channel* $reset_Y$. This mechanism replaces at reset time for $rep(Y)$ all permutations of transitions of simple edges wrt. Y possible in $\mathcal{N}$ by one single transition in $\mathcal{N}'$. That single transition is used if at least one transformed automaton assumes the origin location of a simple edge, i.e. transformed simple edges wrt. Y indicate the reset time for $rep(Y)$. Moreover, since simple edges are taken independently from all other edges, this allows us to take all transformed simple edges in $\mathcal{N}'$ before the first transformed complex one, which in turn allows us to support all queries which ask for configurations where some complex edges and none, only some, or all simple edges have been taken. We enforce this prioritisation by setting the priorities of all channels $reset_Y$ higher than any other channel existing in the original network, including local transitions.

   In this mechanism any transformed automaton with transformed simple edges wrt. Y is able to send and listen on channel $reset_Y$. To this end, we equip each transformed simple edge with an output on $reset_Y$. A copy of that edge is made where the output is replaced by an input on the same channel. Following this mechanism the corresponding $\mathcal{R}_Y$ is equipped with an edge where the resetter listens synchronisations on $reset_Y$. A synchronisation on $reset_Y$ involves the transformed simple edge if and only if the corresponding simple edge would be enabled in $\mathcal{N}$. There are two conditions in $\mathcal{N}'$ for which a simple edge would not be enabled in $\mathcal{N}$ at reset time for $rep(Y)$: (1) the underlying automaton $\mathcal{A}_i$ has reached the origin location of that edge and the time for taking that edge has not been reached yet (as indicated by $s_Y^{\mathcal{A}_i} = 1$ and $t_{x_i} = 0$), and trivially, (2) the underlying automaton $\mathcal{A}_i$ has not reached the origin location of that edge yet (as indicated by $s_Y^{\mathcal{A}_i} = 0$).

   The restrictions on simple edges from Definition 22, e.g. pre/post delay, no communication on channels, only one outgoing edge from the

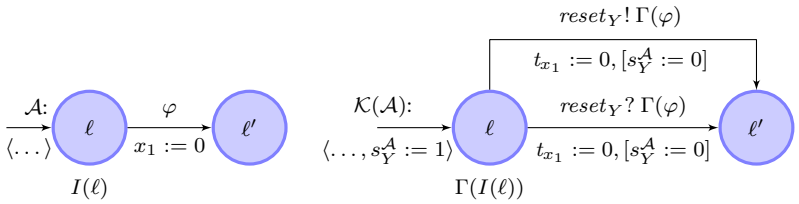
origin location, etc., guarantee that at reset time for $rep(Y)$ a pre-delay greater than 0 has been consumed by all enabled transformed simple edges, and that they send and listen on $reset_Y$. Since multiple automata may have an edge synchronising on $reset_Y$ enabled, there is a slight verification time overhead for checking this enabledness, but all edges induce the exact same follow-up configuration where $\mathcal{R}_Y$ reaches ℓ_{nstY}, and all enabled transformed edges reach their respective destination location.

2. *Urgent broadcast channel u_Y.* For the case that no transformed simple edge is ready to indicate the reset time for $rep(Y)$, the $\mathcal{R}_Y$ also (indirectly) observes whether transformed complex edges wrt. Y are taken. If the first transformed complex edge is taken at reset time, then the sum of tokens wrt. Y will decrease. The resetter $\mathcal{R}_Y$ uses the urgent broadcast channel u_Y in order to transit to $\ell_{nst\mathcal{R}_Y}$ as soon as the sum of tokens wrt. Y is 0, i.e. all enabled transformed resetting edges have been taken, and before a delay greater than 0 time units can occur in the transformed network. By transiting to the *urgent* location $\ell_{nst\mathcal{R}_Y}$, it is ensured that no time elapses unless a configuration corresponding to stability wrt. Y is reached.

3. *Variable $prio_Y$ and broadcast channel $return$.* We use a broadcast channel $return$ to implement a similar mechanism to the one implemented with the broadcast channel $reset_Y$, with the aim of avoiding interleavings among resetters when they are located in their respective $\ell_{nst\mathcal{R}_Y}$ locations, and at the same time point they transit back to their respective initial locations $\ell_{ini\mathcal{R}_Y}$.

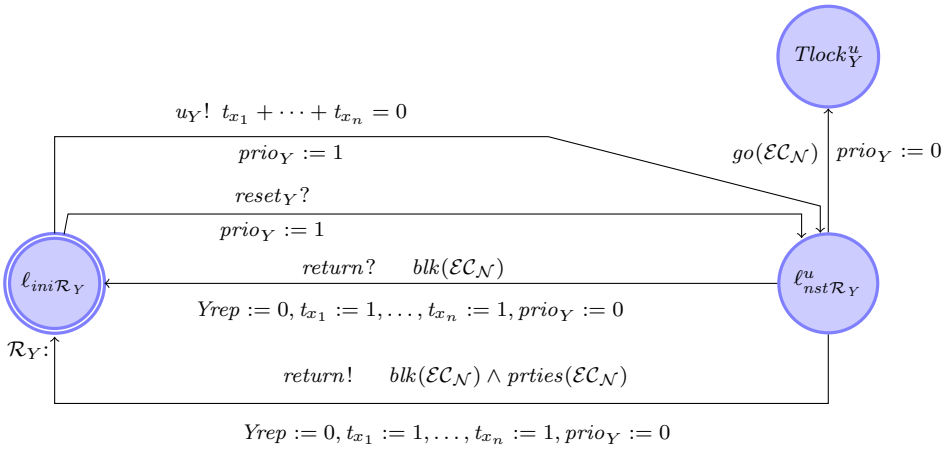
   In this mechanism any resetter is able to send and listen on the channel $return$, and this mechanism is used if: (1) at least resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$, (2) the sum of tokens wrt. Y is 0 and, (3) all other equivalence classes different from Y are stable as expressed by the function $blk(\mathcal{EC}_\mathcal{N})$. As part of this mechanism we equip an edge of $\mathcal{R}_Y$ from $\ell_{nst\mathcal{R}_Y}$ to $\ell_{ini\mathcal{R}_Y}$ resetting the representative clock of Y and updating values of tokens related to clocks in Y to 1, with an output on $return$. A copy of that edge is made where the output is replaced by an input on the same channel. Since multiple resetters may have an edge with an output on $return$ enabled, we eliminate the verification time overhead for checking this enabledness by using the function $prties(\mathcal{EC}_\mathcal{N})$, so that the resetter with the highest priority for its equivalence class at reset time sends on $return$. The synchronisation on $return$ allow all participating resetters to reach $\ell_{ini\mathcal{R}_Y}$ in parallel.

Transformed simple edges wrt. multiple equivalence classes may be ready to indicate at the same time the reset time for the respective representative clocks, or multiple edges in resetters with output on u_Y may be enabled. We avoid interleavings among those edges by assigning priorities on channels u_Y and $reset_Y$ as follows. For all $Y \in \mathcal{EC}_\mathcal{N}$, channels $reset_Y$ and u_Y from the same Y have the same priority. Among equivalence classes from $\mathcal{EC}_\mathcal{N}$, the priorities for those two kinds of channels follow the order given in $\mathcal{EC}_\mathcal{N}^{prio}$ by the user of Algorithm $\mathcal{K}$. All other channels including $return$ as well as local transitions, have the same priority which is smaller than the priority assigned to channels $reset_Y$ and u_Y.



(a) Pattern of a simple edge wrt. Y in automaton $\mathcal{A}$.

(b) Pattern of the simple edge of $\mathcal{A}$ transformed in $\mathcal{K}(\mathcal{A})$.



(c) Resetter $\mathcal{R}_Y$.

FIGURE 5.2: Patterns used to transform a network $\mathcal{N}$ with $\mathcal{EC}_\mathcal{N}$. In figures (a), (b) and (c) we consider the following quasi-equal clocks $Y = \{x_1, \dots, x_n\}$, where $Y \in \mathcal{EC}_\mathcal{N}$. Urgent locations are denoted with the superscript u in the name of those locations. For simplicity in figure (c) we omit drawing the implicit invariants of both urgent locations, and resets to clocks used in these invariants. In figure (c) we use the fresh clock $Yrep$ as the representative clock of Y. In figure (b) the expression $s_Y^{\mathcal{A}} := 0$ in brackets is added if and only if ℓ' is not a reset location of a simple edge.

Following [9] our transformation is able to: (1) transform corner cases of networks with quasi-equal clocks where time may be stopped either at origin or at destination locations of simple edges. To this end, $\mathcal{R}_Y$ checks after synchronising on $reset_Y$ whether a transformed automaton $\mathcal{A}_i$ did not take the synchronisation. The latter is indicated by leaving unchanged the values of t_{x_i} and $s_Y^{\mathcal{A}_i}$, for some $i \in \{1, \dots, q\}$. Therefore, $\mathcal{R}_Y$ transits from $\ell_{nst\mathcal{R}_Y}$ to the urgent location $Tlock_Y$ (cf. Figure 5.2.(c)), where time is stopped and no delay transitions greater than 0 are possible, if and only if the following conditions denoted by the guard $go(\mathcal{EC}_\mathcal{N})$ of the edge incoming to $Tlock_Y$ hold: (a) automaton $\mathcal{A}_i$ leaves unchanged the values of t_{x_i} and $s_Y^{\mathcal{A}_i}$ and, (b) resetters with higher priorities than $\mathcal{R}_Y$ have returned to their respective initial locations. And, (2) provide a direct detection, by reaching $Tlock_Y$, of timelocks in the sense above mentioned, since this detection is at least not direct and more involved in Uppaal.

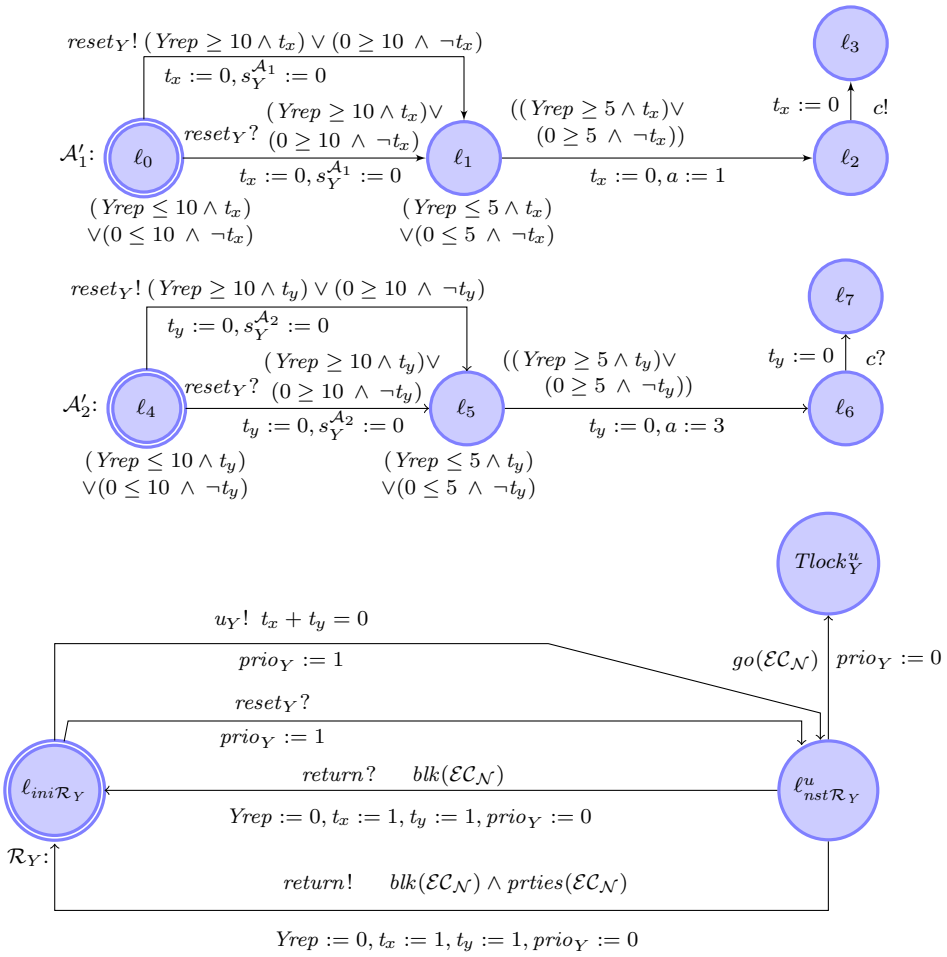


FIGURE 5.3: Transformed network $\mathcal{N}_1' = \mathcal{K}(\mathcal{N}_1, \mathcal{EC}_\mathcal{N}^{prio})$.

Example 4 (Transformation of Network $\mathcal{N}_1$) *Applying $\mathcal{K}$ to network $\mathcal{N}_1$ from Figure 5.1 yields $\mathcal{N}_1'$ (cf. Figure 5.3). Similar to the algorithms in [7, 8, 9], only the representative clock of each equivalence class remains, in our example we use the fresh clock $Yrep$ as representative of Y which is reset by resetter $\mathcal{R}_Y$. Note that each guard and invariant in automata $\mathcal{A}_1'$ and $\mathcal{A}_2'$ is transformed by Γ into a disjunction of clauses. For instance, the guard $x \geq 10$ of automaton $\mathcal{A}_1$ in $\mathcal{N}_1$, is transformed in $\mathcal{N}_1'$ into the disjunction $(Yrep \geq 10 \wedge t_x) \vee (0 \geq 10 \wedge \neg t_x)$. Then the clause $(Yrep \geq 10 \wedge t_x)$ is effective in configurations in $\mathcal{N}'$ where t_x is 1 (encoding that clock x has the same value as $Yrep$), while the clause $(0 \geq 10 \wedge \neg t_x)$ is effective in configurations where t_x is 0 (encoding that x has already been reset and thus has value 0).*

Note that in $\mathcal{N}_1'$ the pair of transformed complex edges from locations ℓ_1 and ℓ_5 preserve their original interleavings. Furthermore, the other pair of transformed complex edges, outgoing from locations ℓ_2 and ℓ_6, are taken simultaneously even without delaying at their origin locations. ♢

5.5 Transforming Queries over Networks

The following function Ω syntactically transforms properties over a network $\mathcal{N}$ with a list of all equivalence classes of quasi-equal clocks of $\mathcal{N}$, $\mathcal{EC}_{\mathcal{N}}^{prio}$, into properties over $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$.

Definition 28 (Function Ω)
Let $\mathcal{N} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}} = \{Y_1, \ldots, Y_m\}$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements of $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let β be a basic formula over $\mathcal{N}$. Let $\ell_{nst\mathcal{R}_Y}$ be the location of resetter $\mathcal{R}_Y$ which encodes unstable configurations wrt. $Y \in \mathcal{EC}_{\mathcal{N}}$. Let $p = |\mathcal{X}(\mathcal{N})|$. For simplicity below, from a basic formula $\mathcal{A}_i.\ell$ we omit the reference to $\mathcal{A}_i$. We define the function Ω as follows:
$\Omega_0(\beta) =$

$$\begin{cases} (\ell' \wedge \tilde{x}) \vee \ell & \text{, if } \beta = \ell, (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_{Y_j}(\mathcal{A}_i). \\ (\ell' \wedge \neg\tilde{x}) & \text{, if } \beta = \ell', (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_{Y_j}(\mathcal{A}_i). \\ \beta & \text{, if } \beta \in \{\ell, \ell'\}, (\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{A}_i) \setminus SimpEdges_{Y_j}(\mathcal{A}_i). \\ \Gamma(\beta)[t_x/(t_x \vee \tilde{x}) \mid x \in Y, Y \in \mathcal{EC}_{\mathcal{N}}] & \\ & \text{, if } \beta = \varphi_{clk} \wedge \varphi_{int}. \end{cases}$$

$$\Omega(CF) = \exists\, \tilde{x}_1, \ldots, \tilde{x}_p \bullet \Omega_0(CF) \wedge \kappa_{\mathcal{N}},$$

where:

$$\kappa_{\mathcal{N}} := \bigwedge_{\substack{1 \le i \le n, \\ 1 \le j \le m, \\ (\ell,\alpha,\varphi,\langle x:=0\rangle,\ell') \in SimpEdges_{Y_j}(\mathcal{A}_i)}} \kappa(x), \qquad \kappa(x) : (\tilde{x} \implies \bigvee_{(\ell,\alpha,\varphi,\langle x:=0\rangle,\ell') \in SimpEdges_{Y_j}(\mathcal{A}_i)} \ell' \wedge \ell_{nst\mathcal{R}_{Y_j}}).$$

By structural induction Ω_0 transforms configuration formulas CF. $\diamondsuit$

Function Ω takes a query over a network $\mathcal{N}$ with quasi-equal clocks, and outputs an equivalent query that can be verified in the corresponding transformed network $\mathcal{N}'$.

We explain the idea behind the transformation of queries carried out by Function Ω. Recall that by Algorithm $\mathcal{K}$ in network $\mathcal{N}'$ we have replaced all clocks in Y by a representative clock $rep(Y)$. Recall that $\mathcal{N}'$ implements a mechanism such that at reset time for $rep(Y)$ all enabled transformed simple edges wrt. Y are taken at once. All destination locations of transformed simple edges are reached together with the location $\ell_{nst\mathcal{R}_Y}$ of resetter $\mathcal{R}_Y$. Thus, all configurations induced in $\mathcal{N}$ by all possible permutations of resets of quasi-equal clocks carried out by simple edges, are encoded in $\mathcal{N}'$ in configurations where resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$.

We provide query transformations for origin and destination locations of simple edges, and for clock constraints using quasi-equal clocks. These transformations together with consistency checks by means of fresh logical variables, e.g. $\tilde{x}$ for the clock $x \in Y$ reset by a simple edge, allow to query in $\mathcal{N}'$ information from encoded configurations. Since a configuration in $\mathcal{N}$ induced by a complex edge resetting quasi-equal clocks has a corresponding one-to-one configuration in $\mathcal{N}'$, hence, no transformation for queries on origin and destination locations of complex edges is provided.

In the following we illustrate how Ω outputs a query transformation and in particular how consistency checks are used.

Example 5 (Querying Reset Locations) *From automaton $\mathcal{A}_2$ of network $\mathcal{N}_1$ (cf. Figure 5.1) consider the simple edge $e = (\ell_4, \tau, (y \geq 10), \langle y := 0\rangle, \ell_5)$. The query $\exists\Diamond\, \Omega(\phi)$, where $\phi := \mathcal{A}_2.\ell_4$, after some simplifications (included those where we omit the reference to $\mathcal{A}_i$ from a basic formula $\mathcal{A}_i.\ell$) is transformed into*

$$\exists\Diamond\, \exists\, \tilde{y} \in \{0,1\} \bullet ((\ell_5 \wedge \tilde{y}) \vee \ell_4) \wedge (\tilde{y} \implies (\ell_5 \wedge \ell_{nst\mathcal{R}_Y})).$$

Note that $\tilde{y}$ is a logical variable related to clock y. From $\Omega(\phi)$, the clause ℓ_4 is satisfied in configurations of $\mathcal{N}_1'$ which are related (cf. Lemma 2) to stable configurations in $\mathcal{N}_1$ where automaton $\mathcal{A}_2$ is located at ℓ_4 during the pre-delay of 10 time units required at that location. The clause $(\ell_5 \wedge \tilde{y})$ is satisfied in a configuration of $\mathcal{N}_1'$ which is related to the unstable configuration of $\mathcal{N}_1$ where $\mathcal{A}_2$ has reset the clock y while the value of x is still 10. This particular configuration of $\mathcal{N}_1'$ is enforced by the consistency check $\tilde{y} \implies (\ell_5 \wedge \ell_{nst\mathcal{R}_Y})$. ♢

Example 6 (Querying Reset and Reset Successor Locations) *From automaton $\mathcal{A}_2$ of network $\mathcal{N}_1$ consider the simple edge $e = (\ell_4, \tau, (y \geq 10), \langle y := 0\rangle, \ell_5)$ (cf. Figure 5.1). The query $\exists\Diamond\, \Omega(\phi)$, where $\phi := \mathcal{A}_2.\ell_4 \wedge \mathcal{A}_2.\ell_5$, is not satisfied in $\mathcal{N}_1$ for obvious reasons, i.e. an automaton cannot be at two locations in the same configuration. Another purpose of logical variables is to avoid that our query transformations output queries that are satisfiable in transformed networks but unsatisfiable in original ones. To this end, we intentionally produce logically unsatisfiable clauses like the one below.*

After some simplifications $\exists\Diamond\, \Omega(\phi)$ is transformed into

$$\exists\Diamond\, \exists\, \tilde{y} \in \{0,1\} \bullet ((\ell_5 \wedge \tilde{y}) \vee \ell_4) \wedge (\ell_5 \wedge \neg\tilde{y}) \wedge (\tilde{y} \implies (\ell_5 \wedge \ell_{nst\mathcal{R}_Y})).$$

From $\Omega(\phi)$, the clause $\ell_4 \wedge (\ell_5 \wedge \tilde{y})$ is not satisfied in $\mathcal{N}_1'$ for obvious reasons, and the clause $(\ell_5 \wedge \tilde{y}) \wedge (\ell_5 \wedge \neg\tilde{y})$ intentionally includes $\tilde{y}$ and $\neg\tilde{y}$ in order to make it logically unsatisfiable. ♢

Chapter 6

Weak Bisimulation

6.1 Introduction

In order to prove our approach correct we establish a weak bisimulation relation between a network with quasi-equal clocks and its respective network transformed with Algorithm $\mathcal{K}$. To this end, we carry out an analysis of the configurations preserved in transformed networks, and we propose a function that relates those configurations to the ones in original networks.

In this chapter we also show that properties wrt. an original network are fully preserved in the transformed network, i.e. the transformed network satisfies a transformed property if and only if the original network satisfies the original property.

6.2 Weak Bisimulation

In the following we introduce Function QE which relates configurations of original networks to configurations of transformed networks. Given a network $\mathcal{N}$ with quasi-equal clocks and its respective transformed network $\mathcal{N}'$, Definition 29 (Function QE) is used to relate pairs of configurations $\dot{s} \in Conf(\mathcal{N})$ and $r \in Conf(\mathcal{N}')$ from an original and its corresponding transformed network, respectively, by: (1) relating values of integer and clock variables existing in both configurations, (2) relating location vectors of both configurations and, (3) enforcing consistency conditions for configurations of transformed networks that we use for relating purposes.

Definition 29 (Function QE) *Let $\mathcal{N}$ be a network. Let $\mathcal{EC}_{\mathcal{N}}$ be a set of equivalence classes of quasi-equal clocks of $\mathcal{N}$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements of $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let $QE : Conf(\mathcal{N}) \rightarrow 2^{Conf(\mathcal{N}')}$ be the function defined pointwise as follows,*

$$QE(\langle \vec{\ell}_{\dot{s}}, \nu_{\dot{s}} \rangle) = \Big\{ r = \langle \{\ell_{\dot{s},1}, \ldots, \ell_{\dot{s},n}, \ell_{\mathcal{R}_{Y_1}}, \ldots, \ell_{\mathcal{R}_{Y_m}}\}, \nu_r \rangle \mid$$

$$(\forall\, x \in V(\mathcal{N}) \bullet \nu_r(x) = \nu_{\dot{s}}(x)) \tag{6.2.1}$$

$$\wedge\ \forall 1 \leq i \leq n\ \bullet \tag{6.2.2}$$

$$\Big(\Big(\ell_{r,i} = \ell_{\dot{s},i} \wedge \forall\, x \in \mathcal{X}(\mathcal{A}_i) \bullet \nu_{\dot{s}}(x) = \nu_r(rep(x)) \cdot \nu_r(t_x) \Big) \tag{6.0.2a}$$

$$\vee\ \Big(\exists\, (\ell, \alpha, \varphi, \langle x := 0 \rangle, \ell') \in SimpEdges_Y(\mathcal{A}_i) \bullet \ell_{\mathcal{R}_Y} \neq \ell_{ini\mathcal{R}_Y} \wedge$$
$$\ell_{\dot{s},i} = \ell\ \wedge \ell_{r,i} = \ell' \wedge \nu_{\dot{s}}(x) = \nu_r(rep(x)) \wedge \nu_r(t_x) = 0 \wedge$$
$$\forall\, y \in \mathcal{X}(\mathcal{A}_i) \setminus \{x\} \bullet \nu_{\dot{s}}(y) = \nu_r(rep(y)) \cdot \nu_r(t_y) \Big) \Big) \tag{6.0.2b}$$

$$\wedge\, \forall\, Y \in \mathcal{EC}_{\mathcal{N}} \bullet$$

$$\Big((\nu_r(s_Y^{\mathcal{A}_i}) = 1 \iff \exists\, (\ell, \alpha, \varphi, \vec{r}, \ell') \in SimpEdges_Y(\mathcal{A}_i) \bullet \ell_{r,i} = \ell) \tag{6.2.3}$$

$$\wedge\, \nu_r(prio_Y) = 1 \iff (\ell_{r,\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y}) \Big) \Big\} \tag{6.2.4}$$

$\diamondsuit$

In the following we explain how Function QE is used. Since by Definition 27 of algorithm $\mathcal{K}$, integer variables are not transformed, hence all original integer variables are used in transformed networks, and their values in both configurations can be directly related (see Rule 6.2.1).

The transformation of resetting edges by algorithm $\mathcal{K}$ has a twofold effect in configurations of transformed networks. On the one hand, we are able to establish an one-to-one relation between configurations of both networks induced by taking delays, or by taking non-simple edges in the original network and non-transformed-simple edges in the transformed network. For this relation we directly relate location vectors, and we also relate values of clocks of the original network with values encoded in transformed networks for those clocks (see Rule 6.0.2a).

On the other hand, we are also able to establish a one-to-many relation between configurations induced: by taking simple edges one by one in the original network, and by taking transformed simple edges in parallel in the transformed network. A configuration of the transformed network where the resetter $\mathcal{R}_Y$, for some $Y \in \mathcal{EC}_{\mathcal{N}}$, is located at ℓ_{nstY} encodes multiple configurations of the original network that are induced by taking simple edges. To establish this one-to-many relation we have Rule (6.0.2b), which considers the locations of resetters and automata in both networks wrt. the origin and destination of simple edges, and values of clocks being reset by these edges.

The following two rules ensure that we relate to each $\dot{s}$ only configurations r whose values for book-keeping and priority variables introduced by our transformation are consistent. For instance, rule 6.2.3 establishes a consistent counter of the number of automata currently located at origin locations of simple edges, while rule 6.2.4 establishes a consistent counter of the number of resetters currently located at their ℓ_{nstY} locations.

We have observed that during stability phases, i.e. transitions between stable configurations, there is a strong bisimulation (one-to-one) between original and transformed networks. Only during unstability phases there is a weak bisimulation (one-to-many) in both directions. There are cases (resets of clocks carried out by simple edges) where original networks simulate one step of transformed networks with multiple steps, and cases (resets clocks carried out by complex edges) where transformed networks simulate one step of original networks with multiple steps. Using these observations we introduce the following lemma.

Lemma 2 *Weak Bisimulation.*
Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements of $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{CF}_{\mathcal{N}}$ be the set of configuration formulas over $\mathcal{N}$. Then $\mathcal{N}$ is *weakly bisimilar* to $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$, i.e. there is a *weak bisimulation relation* $QE \subseteq Conf(\mathcal{N}) \times Conf(\mathcal{N}')$ such that:

1. $\forall\, s \in \mathcal{C}_{ini}(\mathcal{N})\ \exists\, r \in \mathcal{C}_{ini}(\mathcal{N}') \bullet r \in QE(s)$ and $\forall\, r \in \mathcal{C}_{ini}(\mathcal{N}')\ \exists\, s \in \mathcal{C}_{ini}(\mathcal{N}) \bullet r \in QE(s)$.
2. $\forall\, CF \in \mathcal{CF}_{\mathcal{N}} \forall\, r \in QE(s) \bullet s \models_{\delta} CF \implies r \models_{\delta} \Omega(CF)$.
3. $\forall\, CF \in \mathcal{CF}_{\mathcal{N}} \forall\, r \in QE(s) \bullet r \models_{\delta} \Omega(CF) \implies \exists\, \dot{s} \in Conf(\mathcal{N}) \bullet r \in QE(\dot{s}) \wedge \dot{s} \models_{\delta} CF$.
4. $\forall\, r \in QE(s) \bullet s \xrightarrow{\lambda} s' \implies \exists\, r \xrightarrow{\lambda}{}^{*}\, r' \bullet r' \in QE(s')$. A transition $s \xrightarrow{\lambda} s'$ is distinguished as follows:
   (a) Transition is justified by a delay $\lambda = d \geq 0$.
   (b) $s \in \bigcap_{Y \in \mathcal{EC}_{\mathcal{N}}} StableConf_Y, s' \notin \bigcap_{Y \in \mathcal{EC}_{\mathcal{N}}} StableConf_Y$, and the transition is justified by a simple edge wrt. $Y \in \mathcal{EC}_{\mathcal{N}}$.
   (c) $s \notin \bigcap_{Y \in \mathcal{EC}_{\mathcal{N}}} StableConf_Y$ and the transition is justified by a simple edge wrt. $Y \in \mathcal{EC}_{\mathcal{N}}$.
   (d) Transition is justified by a non-empty set of edges E, such that none of them is simple wrt. any $Y \in \mathcal{EC}_{\mathcal{N}}$.
5. $\forall\, CF \in \mathcal{CF}_{\mathcal{N}} \forall\, r \in QE(s) \bullet r \xrightarrow{\lambda} r' \wedge r' \models_{\delta'} \Omega_0(CF) \implies \exists\, s \xrightarrow{\lambda}{}^{*}\, s' \bullet r' \in QE(s')$. A transition $r \xrightarrow{\lambda} r'$ is distinguished as follows:
   (a) transition is justified by a delay $\lambda = d \geq 0$,
   (b) transition is justified by a non-empty set of edges E.

Where $r \xrightarrow{\tau}{}^{*}\, r'$ denotes zero or more successive τ-transitions from configuration r to configuration r'.

The proof of Lemma 2 together with auxiliary material used in the proof is available in Appendix A.

The following theorem shows that properties wrt. an original network with quasi-equal clocks are fully preserved in the transformed network, i.e. the transformed network satisfies a transformed property if and only if the original network satisfies the original property. In other words, in this theorem we show that we construct correct networks where we are able to verify all properties reflected by the original versions.

Theorem 1 *Satisfiability of Queries in $\mathcal{N}'$.*

Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of elements from $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let CF be a configuration formula over $\mathcal{N}$. Then

$$\mathcal{N} \models_{\delta} \exists\Diamond(CF) \iff \mathcal{N}' \models_{\delta} \exists\Diamond\, \Omega(CF).$$

Proof. The proof is straightforward by using Lemma 2, and induction over the length of paths to show that CF holds in $\mathcal{N}$ if and only if $\Omega(CF)$ holds in $\mathcal{N}'$.

Chapter 7

Model Checking on Networks with Quasi-Equal Clocks

7.1 Introduction

In the work of [9] we introduce the theoretical analysis that justifies savings wrt. space and time that are obtained in networks transformed with the version of Algorithm $\mathcal{K}$ from that work.

In that work we propose formulas to quantify the number of reachable configurations in transformed networks, and thereby know the number of configurations saved in those networks. We propose as well worst-case space and time bounds that characterise a less expensive model checking effort in transformed networks wrt. the effort in original networks with quasi-equal clocks.

In this chapter we update that analysis since we introduce improvements in networks transformed with Algorithm $\mathcal{K}$, e.g. a broadcast mechanism which in one transition synchronises transformed simple edges. These improvements lead to savings greater than those obtained in networks transformed with the version of Algorithm $\mathcal{K}$ from [9].

7.2 Zones and Operations on Zones

Recall that Uppaal is the model checker that helps us with the theoretical analysis of the underlying state-space in transformed networks. Some figures used in our analysis, e.g. *states explored*, obtained from the reachability analysis carried out by Uppaal are expressed in terms of zones, therefore, we recall in this chapter the notion of zones and some operations on them.

Definition 30 *Zones*

Given an extended timed automaton $\mathcal{A}_e = (L, B, \mathcal{X}, V, I, E, \ell_{ini})$, a zone $Z \subseteq (\mathcal{X} \cup V \rightarrow Time \cup \mathbb{Z})$ is the maximal set of valuations satisfying a given constraint $\varphi \in \Phi(\mathcal{X}, V)$, i.e. $\nu \in Z \iff \nu \models \varphi$. Then each $\varphi \in \Phi(\mathcal{X}, V)$ is a symbolic representation of a zone[1].
We define the following operations on a zone Z:

- *Time elapse:* $Z^{\uparrow} = \{\nu[\mathcal{X} + t] \mid \nu \in Z \wedge t \in Time\}$, where $\nu[\mathcal{X} + t]$ denotes the valuations obtained from ν by increasing the valuations of all clocks in $\mathcal{X}$ by t time units.

[1]Our notion of zones slightly differs from the one in [11]. In practice the part of a zone related to data values can be explicitly stored, while the part related to clock values symbolically, for instance, in DBMs.

- *Zone intersection:* $Z_1 \wedge Z_2 = \{\nu \mid \nu \in Z_1 \cap Z_2\}$.
- *Zone update:* $Z[\vec{r}\,] = \{\nu[\vec{r}\,] \mid \nu \in Z\}$, where $\nu[\vec{r}\,]$ such that $\vec{r} = \langle Y := 0, v_1 := \phi_1, \ldots, v_n := \phi_n\rangle$ is the reset vector of a given edge, is an abbreviation for $\nu[Y := 0, v_1 := \phi_1, \ldots, v_n := \phi_n]$ which denotes the valuations obtained from ν by resetting all clocks in Y, and setting the variables $v_1, \ldots, v_n$ to the integer values $\phi_1, \ldots, \phi_n$, respectively.

We lift configurations of networks to pairs consisting of a location vector and a zone. For extended timed automata $\mathcal{A}_{e_i} = (L_i, B_i, \mathcal{X}_i, V_i, I_i, E_i, \ell_{ini,i})$ with $i = 1, \ldots, n$, pairwise disjoint set $\mathcal{X}_i$ of clocks, and pairwise disjoint set L_i of locations, consider the network $\mathcal{N} = \{\mathcal{A}_{e_1}, \ldots, \mathcal{A}_{e_n}\}$. The set of configurations of $\mathcal{N}$, i.e. $Conf(\mathcal{N})$, consists of pairs, e.g. $\langle \vec{\ell}, Z\rangle$, of a location vector $\vec{\ell} = \langle \ell_1, \ldots, \ell_n\rangle$ from $\times_{i=1}^{n} L_i$, and a zone $Z = Z' \wedge \bigcup_{1 \leq i \leq n} I_i(\ell_i)$ for some Z'.

Now redefining the semantics of extended timed automata, computations paths, the logic of Uppaal, and any other definition from the previous chapters where we have used configurations in the sense of pairs consisting of a location vector and a valuation (see Definition 16 of Chapter 3), is straightforward by using Definition 30, however, we will not do this, and from now on we will just assume that the semantics of extended timed automata, computations paths, the logic of Uppaal, etc., use configurations in the sense of the above paragraph.

7.3 Savings in Transformed Networks

Recall the two principal facts that justify the savings wrt. space (in number of configurations) and time obtained in networks transformed with Algorithm $\mathcal{K}$, whose original versions are networks where all quasi-equal clocks are exclusively reset by simple edges:

1. The size of a set of reachable configurations induced in original networks by executing all possible permutations of resets of quasi-equal clocks, increases exponentially in the number of quasi-equal clocks reset. Replacing these permutations in transformed networks by a broadcast transition and encoding these configurations, lead to a drastic reduction in the number of configurations that Uppaal explores when checking properties in transformed networks.

2. A reduction in the size of DBMs, which is yielded by using only the representative clocks of each equivalence class in order to represent zones in Uppaal, leads to a more efficient representation, storage, access and manipulation of DBMs in memory.

In this section we exploit the first fact, and propose a formula to quantify the number of reachable configurations in those transformed networks, and thereby know the number of configurations saved in those networks. Moreover, we show that those transformed networks have a smaller state-space as compared to the state-space of the original versions. Now we present the notion for *reset configurations* which are essential to our quantification of savings, since we use those configurations to calculate the number of reachable configurations that we encode in transformed networks.

Definition 31 (Reset Configuration)
Let $\mathcal{N}$ be a network, and $Y \in \mathcal{EC}_{\mathcal{N}}$ be a set of quasi-equal clocks of $\mathcal{N}$. Let $s, s' \in \mathit{Conf}(\mathcal{N})$ be two reachable configurations of $\mathcal{N}$. Then s is called *reset configuration wrt. Y*, if and only if s is stable wrt. Y, and if some edges $e_1, \ldots, e_k \in \mathit{SimCompEdges}_Y(\mathcal{N})$ justify a transition from s to s', i.e. if $\exists\, s_0 \xrightarrow{\lambda_1}_{E_1} s_1 \ldots s_{n-1} \xrightarrow{\lambda_n}_{E_n} s_n \in \mathit{Paths}(\mathcal{N}) \bullet e_1, \ldots, e_k \in E_n \wedge s_{n-1} \in \mathit{StableConf}_Y \wedge s = s_{n-1} \wedge s' = s_n$. We write $s_i \xrightarrow{\lambda_i}_{E_i} s_{i+1}$, $i \in \mathbb{N}$, to denote that the (non-delay) transition $s_i \xrightarrow{\lambda_i} s_{i+1}$ is justified by the set of edges E_i; E_i is empty for delay transitions, i.e. if $\lambda_i \in \mathit{Time}$. ♢

The set $\mathit{ResetConf}_Y$ contains all reset configurations of $\mathcal{N}$ wrt. $Y \in \mathcal{EC}_{\mathcal{N}}$, and $RC = \bigcup_{Y \in \mathcal{EC}_{\mathcal{N}}} \mathit{ResetConf}_Y$ contains all reset configurations wrt. each Y. For a given reset configuration $s \in \mathit{ResetConf}_Y$, $\mathit{clks}(s) = \{x \in \mathcal{X}(\mathcal{N}) \mid \exists Y \in \mathcal{EC}_{\mathcal{N}} \bullet s \in \mathit{ResetConf}_Y \wedge x \in Y\}$ obtains the quasi-equal clocks to be reset from s, and $\mathit{class}(s) = \{Y \in \mathcal{EC}_{\mathcal{N}} \mid s \in \mathit{ResetConf}_Y\}$ obtains the equivalence classes of those clocks.

Observe the following: in a network $\mathcal{N}$ where all quasi-equal clocks are exclusively reset by simple edges, at a given time point a number of reachable configurations is induced by executing all possible permutations of resets of those clocks. We are able to precisely calculate that number since:

1. Once that an automaton $\mathcal{A}$ reaches the origin location ℓ of a simple edge e, which is the only outgoing edge from ℓ, $\mathcal{A}$ must carry out a pre-delay $d > 0$ at ℓ before resetting a quasi-equal clock x with e.
2. A post-delay $d' > 0$ enforced at the destination location ℓ' of e, hinders carrying out action transitions in zero time after resetting x.

Hence, at time points where x is reset we know that $\mathcal{A}$ is either located at ℓ and x has the same value as $\mathit{rep}(x)$, or located at ℓ' and the value of x is 0.

Observe that in a network transformed with Algorithm $\mathcal{K}$ (Definition 27) we remove all quasi-equal clocks but representative ones, and all reachable configurations (in the respective original network) induced by all possible permutations of resets of quasi-equal clocks, are encoded in the transformed network by configurations induced by broadcast synchronisations between resetters and automata with transformed simple edges.

The version of Algorithm $\mathcal{K}$ from [9] constructs a fixed sequence of transitions that represents all those permutations. In this work Algorithm $\mathcal{K}$ replaces that sequence by one transition to obtain greater savings.

Considering these observations we present a formula to calculate the number of reachable configurations of transformed networks, and thereby know the number of configurations saved.

Lemma 3 ($\mathit{SimpleEdge}-$Formula)
Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, where all quasi-equal clocks are exclusively reset by simple edges. Let $\mathcal{EC}^{prio}_{\mathcal{N}}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_{\mathcal{N}})$. Then the following formula yields the number of reachable configurations in $\mathcal{N}'$:

$$|\mathit{Reach}_{\mathcal{N}}| - \Big(\sum_{s \in RC} 2^{|\mathit{clks}(s)|} \Big) + \sum_{s \in RC} \Big[|\mathit{class}(s)| + 2 \Big].$$

Proof. Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, where each clock $x \in Y$, $Y \in \mathcal{EC}_{\mathcal{N}}$, is exclusively reset by simple edges in $SimpEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. We calculate the number of reachable configurations of $\mathcal{N}'$ as follows:

- The term $|Reach_{\mathcal{N}}|$, which can be obtained by verifying $\mathcal{N}$ in Uppaal, is the number of reachable configurations of $\mathcal{N}$ which includes all reachable configurations induced by all possible permutations of resets of quasi-equal clocks.
- Considering a single clock $x \in Y$ and a reset configuration $s \in RC$, we obtain 2 intervals for the values of x, i.e. the value (in s) just before and the value (in s') just after resetting x by a simple edge. These intervals can be characterised by the following constraints:

  $$[x = rep(x)], [x = 0].$$

  Considering all quasi-equal clocks, and all reset configurations in RC, thus together yields $\sum_{s \in RC} 2^{|clks(s)|}$, which is the number of reachable configurations induced by resetting all clocks in each set $clks(s)$. These configurations exist in $\mathcal{N}$ but are encoded in $\mathcal{N}'$.
- Considering a reset configuration $s \in RC$, the term $|class(s)| + 2$ is the number of configurations that at a given time point are induced in $\mathcal{N}'$, by a fixed sequence of transitions that represents all possible permutations in $\mathcal{N}$ of resets of quasi-equal clocks wrt. all equivalence classes in $class(s)$. Then the term is calculated as follows:
  - The subterm $|class(s)|$ yields the number of equivalence classes for which s is a reset configuration. This number is equal to the number of configurations induced by taking in $\mathcal{N}'$ a sequence of broadcast transitions on the channel $reset_Y$, with $Y \in class$, where all transformed simple edges wrt. Y enabled in a respective configuration $r \in Conf(\mathcal{N}')$, with $r \in QE(s)$, participate (point (b) of case 4 in Lemma 2 confirms this number of transitions in $\mathcal{N}'$). Note that each resetter $\mathcal{R}_Y$ also participates and transits to $\ell_{nst\mathcal{R}_Y}$ in each broadcast transition on the channel $reset_Y$.
  - The subterm 2 is the number of configurations comprising the starting configuration of the sequence of broadcast transitions, and the configuration reached when each $\mathcal{R}_Y$ transits to $\ell_{ini\mathcal{R}_Y}$. These configurations are induced when $\mathcal{R}_Y$ participates in a broadcast transition on the channel $return$ (points (a) and (b) of case 4 in Lemma 2 confirms this number).

  Considering each reset configuration in the set RC together thus yields $\sum_{s \in RC} \Big[|class(s)| + 2 \Big]$ reachable configurations.

This completes our calculations. □

For networks with a set of equivalence classes of quasi-equal clocks, where all those clocks are exclusively reset by simple edges, we distinguish cases when the number of reachable configurations of original networks is smaller than the one in their respective transformed networks. These cases occur when some of those equivalence classes consist of only one clock. Then in original networks there is one reachable configuration induced for each reset of those single clocks, while in transformed networks there are two reachable configurations induced by each fixed sequence of transitions representing those transitions which reset single clocks.

The next lemma states for which transformed network its state-space is strictly smaller than the state-space of the original network. That is, we claim that the number of reachable configurations of a transformed network is strictly smaller than the number of reachable configurations of its respective original version, if the original network has a set of equivalence classes of quasi-equal clocks, where all those clocks are exclusively reset by simple edges, and where each equivalence class consist of at least two clocks.

Lemma 4 (Smaller State-Space)
Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, where $|Y| \geq 2$, for all $Y \in \mathcal{EC}_{\mathcal{N}}$, and where each clock $x \in Y$ is exclusively reset by simple edges in $SimpEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Then $|Reach_{\mathcal{N}'}| < |Reach_{\mathcal{N}}|$.

Proof. Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, where $|Y| \geq 2$, for all $Y \in \mathcal{EC}_{\mathcal{N}}$, and where each clock $x \in Y$ is exclusively reset by simple edges in $SimpEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$.

Since all quasi-equal clocks in $\mathcal{N}$ are exclusively reset by simple edges we can consider the size of the set of all reachable configurations of $\mathcal{N}$ as a sum of two numbers, i.e. $|Reach_{\mathcal{N}}| = X + \big(\sum_{s \in RC} 2^{|clks(s)|}\big)$, where $\big(\sum_{s \in RC} 2^{|clks(s)|}\big)$ is the number of reachable configurations induced by all possible permutations of resets of quasi-equal clocks, and X is the number of all other reachable configurations of $\mathcal{N}$.

By Lemma 3, $|Reach_{\mathcal{N}'}| = |Reach_{\mathcal{N}}| - \big(\sum_{s \in RC} 2^{|clks(s)|}\big) + \sum_{s \in RC} \Big[|class(s)| + 2\Big]$.

Given that $|Reach_{\mathcal{N}}| = X + \big(\sum_{s \in RC} 2^{|clks(s)|}\big)$. Then

$$|Reach_{\mathcal{N}'}| = X + \big(\sum_{s \in RC} 2^{|clks(s)|}\big) - \big(\sum_{s \in RC} 2^{|clks(s)|}\big) + \sum_{s \in RC} \Big[|class(s)| + 2\Big] = X + \sum_{s \in RC} \Big[|class(s)| + 2\Big].$$

By $|Y| \geq 2$, for all $Y \in \mathcal{EC}_{\mathcal{N}}$, we have $\sum_{s \in RC} |clks(s)| \geq \sum_{s \in RC} |class(s)|$. Hence,

$$X + \sum_{s \in RC} 2^{|clks(s)|} \geq X + \sum_{s \in RC} 2^{|class(s)|} > X + \sum_{s \in RC} \Big[|class(s)| + 2\Big].$$

Thus, $$|Reach_{\mathcal{N}'}| < |Reach_{\mathcal{N}}|.$$ □

7.3.1 Pathological Cases

The formula from Lemma 3 precisely yields the number of reachable configurations of transformed networks. However, there are two pathological cases for which the formula yields only a lower bound:

1. Networks where all quasi-equal clocks are exclusively reset by simple edges, and where time is stopped either at origin or at destination locations of those edges. In those networks some simple edges may not be taken and they may not reset quasi-equal clocks. However, the $SimpleEdge-$Formula would be "blind" since its second term would wrongly calculate the number of configurations induced by all possible permutations of resets of simple edges.

   Checking that time is not stopped as above mentioned could require model checking the network, which can be expensive. However, an inexpensive syntactic check can be the following, if an edge has a clock constraint which differs with guards of simple edges only in the constrained clock, and the destination location of that edge is urgent or committed with no outgoing edges, then is very likely that time is stopped as above mentioned.

2. Uppaal uses the *widening operator* [23, 45] on zones to enforce termination of the forward reachability analysis performed by the tool. This operator makes zones even coarser and helps to subsume configurations during that analysis.

   The following networks are examples where using the widening operator helps to subsume configurations during the forward reachability analysis. Networks where: (1) all quasi-equal clocks are reset by only simple edges, (2) simple edges wrt. two or more equivalence classes have in common the same constant occurring in their clock guards and, (3) there is a "loop" through a non-looped edge incoming to the origin location of these simple edges.

   Configurations induced through the transit of those "loops" which without the widening operator would differ only in their underlying zones, are explored only once during that analysis because they are subsumed. However, those widened configurations have corresponding *subsumable configurations* (cf. Definition 32) in transformed versions of these networks. Subsumable configurations are explored more than once since the widening operator does not help them to be subsumed during the reachability analysis.

   The total number of subsumable configurations is difficult to calculate. The formula from Lemma 3 would yield as well the exact number of reachable configurations of transformed networks with subsumable configurations, if the reachability analysis in Uppaal could be extended to subsume subsumable configurations under already explored configurations.

Note that none of our benchmarks (see Chapter 8) are pathological cases.

Definition 32 (Subsumable Configuration)
Let $\mathcal{N}$ be a network with a set $Y \in \mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let $\langle \vec{\ell}_s, Z_s \rangle, \langle \vec{\ell}_{s'}, Z_{s'} \rangle \in Conf(\mathcal{N}')$ be two reachable configurations of $\mathcal{N}'$. Let $(\ell, \tau, x \geq c, \langle x := 0 \rangle, \ell') \in SimpEdges_Y(\mathcal{N})$ be a simple edge.

Then configuration s' is *subsumable* under s if and only if the following conditions hold:

1. The ith automaton (excluding resetters) of $\mathcal{N}'$ is located at the same location in both configurations, i.e. $\forall\, i \in \mathbb{N} \bullet \ell_{s',i} = \ell_{s,i}$.
2. Resetter $\mathcal{R}_Y$ is either located at the same location in both configurations, or in s' is located at ℓ_{iniY} and in s at ℓ_{nstY}.
3. The value of token $t_x, x \in Y$, is either the same in both configurations, or 1 in s' and 0 in s.
4. The value of $prio_Y$ is either the same in both configurations, or 0 in s' and 1 in s.
5. The value of each $s_Y^{\mathcal{A}}, \mathcal{A} \in \mathcal{N}$, is the same in both configurations.
6. The zone $Z_1 = \{\nu|_{\mathcal{X}(\mathcal{N}') \cup V(\mathcal{N})} \mid \nu \in Z_{s'}\}$ is a subset of the zone $Z_2 = \{\nu|_{\mathcal{X}(\mathcal{N}') \cup V(\mathcal{N})} \mid \nu \in Z_s\}$, i.e. $Z_1 \subseteq Z_2$. ♢

7.4 Bounds in Transformed Networks

In most of our experiments with transformed networks which contain transformed complex edges we are only able to deliver savings wrt. time due to the reduced size of DBMs. This reduction leads to a more efficient manipulation of those data structures.

There are two reasons which in combination present a risk of increasing the size of the reachable state-space of a transformed network. The first, reachable configurations of a network induced by resetting quasi-equal clocks with complex edges, have related (one-to-one) reachable configurations in the respective transformed network (see (d) of Lemma 2). The second, is the number of configurations induced by the involved interleaving caused by resetters at time points where representative clocks are reset.

Precisely calculating the size of the reachable state-space in transformed networks which contain transformed complex edges, is very complicated and it would lead to introducing drastic changes in our Algorithm $\mathcal{K}$. We propose an alternative which consists of calculating the bounds where that size lies. Thus, under assumptions on original networks in the following we present lower and upper bounds of the number of reachable configurations of transformed networks with transformed complex edges.

Lemma 5 (Lower Bound $ComplexEdge-$Formula)
Let $\mathcal{N}$ be a network with $\mathcal{EC}_{\mathcal{N}} = \{Y\}$, such that each clock in Y is exclusively reset by complex edges in $CompEdges_Y(\mathcal{N})$, and such that at any time point in which a clock in Y is reset, only edges in $CompEdges_Y(\mathcal{N})$ interleave. Let $\mathcal{EC}^{prio}_{\mathcal{N}}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_{\mathcal{N}})$.

Then $|Reach_{\mathcal{N}}| + (|RC| \cdot 2)$ is a lower bound of the number of reachable configurations of $\mathcal{N}'$.

Proof. Let $\mathcal{N}$ be a network with $\mathcal{EC}_{\mathcal{N}} = \{Y\}$, such that each clock in Y is exclusively reset by complex edges in $CompEdges_Y(\mathcal{N})$, and such that at any time point in which a clock in Y is reset, only edges in $CompEdges_Y(\mathcal{N})$ interleave. Let $\mathcal{EC}^{prio}_{\mathcal{N}}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_{\mathcal{N}})$. We calculate the number of reachable configurations of $\mathcal{N}'$ as follows:

- The term $|Reach_{\mathcal{N}}|$, which can be obtained by model checking $\mathcal{N}$ in Uppaal, is the number of reachable configurations of $\mathcal{N}$ which includes all reachable configurations induced by all possible permutations of resets of quasi-equal clocks. Note that in $\mathcal{N}'$ those permutations and their resulting configurations are preserved, because they are executed by transformed complex edges.

- Recall the following condition, in $\mathcal{N}$ at any time point in which a clock in Y is reset, only edges in $CompEdges_Y(\mathcal{N})$ interleave. This condition is as well preserved in $\mathcal{N}'$ so that when resetter $\mathcal{R}_Y$ performs its transitions no other automaton interleaves. Hence, considering a reset configuration $s \in RC$, there are 2 reachable configurations induced by transitions of $\mathcal{R}_Y$ from and back to its initial location added (see point (a) case 4 in Lemma 2) . Considering each reset configuration in the set RC together thus yields $|RC| \cdot 2$ reachable configurations added.

  This completes our calculations. □

Lemma 6 (Upper Bound $ComplexEdge-$Formula)
Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, such that some clocks in some $Y \in \mathcal{EC}_{\mathcal{N}}$ are reset by some complex edges in $CompEdges_Y(\mathcal{N})$. Note that there might be clocks in some $Y \in \mathcal{EC}_{\mathcal{N}}$ reset by simple edges in $SimpEdges_Y(\mathcal{N})$. Let $\mathcal{EC}^{prio}_{\mathcal{N}}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_{\mathcal{N}})$. Then the number of reachable configurations of $\mathcal{N}'$ is bounded above by $|Reach_{\mathcal{N}}| \cdot \big(\sum_{s \in RC} 2^{|class(s)|}\big)$.

Proof. Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, such that some clocks in some $Y \in \mathcal{EC}_{\mathcal{N}}$ are reset by some complex edges in $CompEdges_Y(\mathcal{N})$. Let $\mathcal{EC}^{prio}_{\mathcal{N}}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_{\mathcal{N}})$.We calculate the number of reachable configurations of $\mathcal{N}'$ as follows:

- The factor $|Reach_{\mathcal{N}}|$, which can be obtained by model checking $\mathcal{N}$ in Uppaal, is the number of reachable configurations of $\mathcal{N}$ which includes all reachable configurations induced by all possible permutations of resets of quasi-equal clocks. Note that in $\mathcal{N}'$ those permutations and their resulting configurations are preserved if they are executed by transformed complex edges.

- Considering a reset configuration $s \in RC$ and an equivalence class $Y \in class(s)$ there are 2 transitions of the resetter $\mathcal{R}_Y$, i.e. from and back to its initial location. These two transitions of $\mathcal{R}_Y$ may interleave with transitions justified by other edges of other automata (see point (a) case 4 in Lemma 2). These two transitions in the worst-case multiplies by factor $|Reach_{\mathcal{N}}|$ the number of reachable configurations of $\mathcal{N}'$. Considering each reset configuration in the set RC, and each equivalence class in the set $class(s)$ together thus yields $\sum_{s \in RC} 2^{|class(s)|}$ transitions which in the worst-case multiplies by factor $|Reach_{\mathcal{N}}|$ the number of reachable configurations of $\mathcal{N}'$.

  This completes our calculations. □

7.5 Complexity of the Model Checking Algorithm on Networks with Quasi-Equal Clocks

The TCTL model checking algorithm from [6] provides us with a convenient framework for discussing in this section worst-case space and time complexity bounds, that characterise a less expensive model checking effort in transformed networks wrt. the effort in original networks with quasi-equal clocks. Recall that the worst-case space complexity of the Timed Computation Tree Logic (TCTL) model checking algorithm (see Algorithm 44 in page 737 of [6]) on a network of time automata, is exponential in the number of clocks in that network [6].

Consider a network $\mathcal{N}$ with quasi-equal clocks where all clocks are exclusively reset by simple edges, and its transformed version $\mathcal{N}'$. The worst-case space complexity of the model checking algorithm on $\mathcal{N}'$ is exponential in the number of equivalence classes of quasi-equal clocks despite the additional modelling elements added to $\mathcal{N}'$, e.g. resetters, boolean flags, etc. That is, the worst-case complexity of the model checking algorithm on $\mathcal{N}'$ lies in an easier complexity class than the worst-case complexity wrt. $\mathcal{N}$.

Recall that zones of an automaton can be broken down into clock regions [3] if the automaton is able to recognise those regions. Therefore, the next bound follows the one for clock regions of timed automata presented and proven in [3], pages 159-160.

Theorem 2

Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}} = \{Y_1, \ldots, Y_m\}$, where each clock $x \in Y$, for each $Y \in \mathcal{EC}_{\mathcal{N}}$, is exclusively reset by simple edges in $SimpEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $c_{rep(Y)} \in \mathbb{N}$ be the maximal constant for each $rep(Y)$ that $\mathcal{N}$ can distinguish, and $c = max\{c_{rep(Y)} \mid Y \in \mathcal{EC}_{\mathcal{N}}\}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$, where $\mathcal{N}' = \{\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{R}_{Y_1}, \ldots, \mathcal{R}_{Y_m}\}$. Then the number of configurations of $\mathcal{N}'$ is bounded above by:

$$|L(\mathcal{A}_1) \times \cdots \times L(\mathcal{A}_n) \times L(\mathcal{R}_{Y_1}) \times \cdots \times L(\mathcal{R}_{Y_m})| \cdot$$
$$(2c+2)^{|\mathcal{EC}_{\mathcal{N}}|} \cdot (4c+3)^{\frac{1}{2}|\mathcal{EC}_{\mathcal{N}}| \cdot (|\mathcal{EC}_{\mathcal{N}}|-1)} \cdot 2^{|V(\mathcal{N})|}.$$

Proof. [*Sketch*]. The size of the state-space of a network of timed automata can be expressed as the multiplication of three numbers, where the first is the number of locations of the network, the second is the number of clock regions [3] of the network, and the third is the number of all possible combinations of values of variables of the network. We consider here for simplicity and without loss of generality only boolean variables.

For $\mathcal{N}'$ the number of locations is at most:

$$|L(\mathcal{A}_1) \times \cdots \times L(\mathcal{A}_n) \times L(\mathcal{R}_{Y_1}) \times \cdots \times L(\mathcal{R}_{Y_m})|.$$

We use the bound of the number of clock regions presented and proven in [3], pages 159-160, which for the case of $\mathcal{N}$ is exponential in the size of its set of clocks $\mathcal{X}(\mathcal{N})$:

$$(2c+2)^{|\mathcal{X}(\mathcal{N})|} \cdot (4c+3)^{\frac{1}{2}|\mathcal{X}(\mathcal{N})| \cdot (|\mathcal{X}(\mathcal{N})|-1)},$$

and we briefly discuss why in the case of $\mathcal{N}'$ that bound is exponential in the size of $\mathcal{EC}_{\mathcal{N}}$.

In $\mathcal{N}$ all quasi-equal clocks are exclusively reset by simple edges, therefore there are configurations induced by all possible permutations of resets of those clocks. In $\mathcal{N}'$ those permutations are replaced by broadcast transitions in which the transitions justified by transformed simple edges are synchronised (see point (b) of case 4 in Lemma 2). Using these broadcast transitions reduces the number of reachable configurations of $\mathcal{N}'$. Hence, despite the additional modelling elements added to $\mathcal{N}'$, e.g. resetters, boolean flags, etc., the size of the state-space of $\mathcal{N}'$ is strictly smaller than the size of the state-space of $\mathcal{N}$ (see Lemma 4).

In $\mathcal{N}'$ there are no configurations which distinguish differences on the values of either quasi-equal clocks of the same equivalence class, or of token t_x-variables related to the same equivalence class. Differences on the values of representative clocks, or on the values of t_x-variables related to the different equivalence classes, can be nevertheless distinguished. Thus, in the case of $\mathcal{N}'$ the number of clock regions is at most:

$$(2c+2)^{|\mathcal{EC}_{\mathcal{N}}|} \cdot (4c+3)^{\frac{1}{2}|\mathcal{EC}_{\mathcal{N}}| \cdot (|\mathcal{EC}_{\mathcal{N}}|-1)}.$$

Recall that in $\mathcal{N}'$ there are only three kinds of variables, i.e. $prio_Y$, $s_Y^{\mathcal{A}}$ and t_x, which do not exist in $\mathcal{N}$. We mentioned that differences on the values of t_x-variables related to the same equivalence class cannot be distinguished. Regarding variables $prio_Y$, $s_Y^{\mathcal{A}}$, their values do not enlarge the state-space of $\mathcal{N}'$, since these variables are mere syntactic sugar for the location in a given configuration of $\mathcal{R}_Y$ and $\mathcal{A} \in \mathcal{N}'$, respectively. Thus, in the case of $\mathcal{N}'$ the number of all possible combinations of values of variables is at most:

$$2^{|V(\mathcal{N})|}.$$

□

Consider a network $\mathcal{N}$ with quasi-equal clocks where some of those clocks are reset by complex edges, and its transformed version $\mathcal{N}'$. Recall that $\mathcal{N}'$ preserves all configurations of $\mathcal{N}$ which are induced by all possible permutations of resets of quasi-equal clocks carried out by complex edges. Therefore, the worst-case space complexity of the TCTL model checking algorithm on $\mathcal{N}'$ is exponential in the number of clocks of $\mathcal{N}$. Thereby, together with the additional configurations induced by interleavings of resetters, the size of the state-space of $\mathcal{N}'$ can be bigger than the one of $\mathcal{N}$.

Theorem 3

Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}} = \{Y_1, \ldots, Y_m\}$, such that for some $Y \in \mathcal{EC}_{\mathcal{N}}$, some clocks in Y are reset by some complex edges in $CompEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $c_{rep(Y)} \in \mathbb{N}$ be the maximal constant for each $rep(Y)$ that $\mathcal{N}$ can distinguish, and $c = max\{c_{rep(Y)} \mid Y \in \mathcal{EC}_{\mathcal{N}}\}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$, where $\mathcal{N}' = \{\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{R}_{Y_1}, \ldots, \mathcal{R}_{Y_m}\}$. Then the number of configurations of $\mathcal{N}'$ is bounded above by

$$|L(\mathcal{A}_1) \times \cdots \times L(\mathcal{A}_n) \times L(\mathcal{R}_{Y_1}) \times \cdots \times L(\mathcal{R}_{Y_m})| \cdot$$
$$(2c+2)^{|\mathcal{X}(\mathcal{N})|} \cdot (4c+3)^{\frac{1}{2}|\mathcal{X}(\mathcal{N})| \cdot (|\mathcal{X}(\mathcal{N})|-1)} \cdot 2^{V(\mathcal{N}')}.$$

Proof. [*Sketch*]. Similar to the proof sketch for Theorem 2. For $\mathcal{N}'$ the number of locations is at most:

$$|L(\mathcal{A}_1) \times \cdots \times L(\mathcal{A}_n) \times L(\mathcal{R}_{Y_1}) \times \cdots \times L(\mathcal{R}_{Y_m})|.$$

Regarding the upper bound of the number of clock regions presented and proven in [3], pages 159-160, for the case of $\mathcal{N}$ is exponential in the size of its set of clocks $\mathcal{X}(\mathcal{N})$, i.e. $(2c+2)^{|\mathcal{X}(\mathcal{N})|} \cdot (4c+3)^{\frac{1}{2}|\mathcal{X}(\mathcal{N})| \cdot (|\mathcal{X}(\mathcal{N})|-1)}$.

In $\mathcal{N}'$ since token t_x-variables related to the same equivalence class are updated by transformed complex edges, and the interleavings of those edges are preserved, therefore, we can distinguish differences on the values of those variables. According to Lemma 2, case 4.(d) the configurations of $\mathcal{N}'$ where those differences can be distinguished, are related one-to-one to those configurations of $\mathcal{N}$ which are induced by all possible permutations of resets of quasi-equal clocks carried out by complex edges. Thus, the same bound of the number of clock regions for the case of $\mathcal{N}$ holds for $\mathcal{N}'$.

By the fact that differences on the values of token t_x-variables related to the same equivalence class can be distinguished in configurations of $\mathcal{N}'$, therefore, the number of all possible combinations of values of variables in $\mathcal{N}'$ is at most:

$$2^{|V(\mathcal{N}')|}.$$

□

With the bounds calculated in Theorem 2 and in Theorem 3 we are able to calculate an upper bound for the worst-case run-time complexity of the TCTL model checking algorithm on networks with equivalence classes of quasi-equal clocks. In particular for the case of transformed networks whose original versions contain quasi-equal clocks exclusively reset by simple edges, the upper bound characterises a less expensive run-time complexity due to a reduced number of reachable configurations of those transformed networks.

The following bounds follow the one presented in [6], page 736.

Theorem 4

Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, where each clock $x \in Y$, for each $Y \in \mathcal{EC}_{\mathcal{N}}$, is exclusively reset by simple edges in $SimpEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let S be the number of configurations obtained in Theorem 2. Let T be the number of transitions which induce at most S configurations of $\mathcal{N}'$. Let ϕ be a query over $\mathcal{N}$ and $\phi' := \Omega(\phi)$. Let ϕ'_{parse} denote the parse tree of ϕ' and $|\phi'_{parse}|$ its size. Then the time complexity of the TCTL model checking algorithm (see Algorithm 44 in page 737 of [6]) on the input $\mathcal{N}'$ is bounded from above by

$$(S + T) \cdot |\phi'_{parse}|.$$

Proof. [*Sketch*]. Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, where each clock $x \in Y$, for each $Y \in \mathcal{EC}_{\mathcal{N}}$, is exclusively reset by simple edges in $SimpEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let ϕ be a query over $\mathcal{N}$ and $\phi' := \Omega(\phi)$. Let ϕ'_{parse} denote the parse tree of ϕ' and $|\phi'_{parse}|$ its size.

The first summand, i.e. S, is the bound for the number of configurations of $\mathcal{N}'$ obtained in Theorem 2. The second summand, i.e. T, is a bound for the number of transitions which induce at most S configurations of $\mathcal{N}'$. The bound T considers the number of transitions induced by: (1) delay transitions and, (2) transitions that combine two steps: firstly time passes and then an internal or synchronization or broadcast transition is taken.

Note that T does not consider a number of transitions representing all possible permutations of resets of quasi-equal clocks justified by taking simple edges, because in $\mathcal{N}'$ we use only representative clocks, and all those possible permutations are replaced by a fixed number of broadcast transitions (see point (b) of case 4 in Lemma 2).

Recall that the mentioned transitions involve operations on zones, e.g. time elapse, zone interception, etc. Recall that zones are represented by DBMs in Uppaal. The size of a DBM for the case of $\mathcal{N}'$ is quadratic in the size of $\mathcal{EC}_{\mathcal{N}}$. Note that these DBMs will induce less operations on them because of their reduced size, as compared to the size of DBMs (quadratic in the number of clocks) used to represent zones of $\mathcal{N}$.

In the following we discuss the factor $|\phi'_{parse}|$. According to [6]: (1) the TCTL model checking algorithm considers path formulas in existential normal form (ENF) and, (2) for each universal path formula there exists an

equivalent path formula in ENF. Then the size of the parse tree of a path formula equals the number of existential path operators located in the nodes of that tree.

Recall that we use the model checker Uppaal and the logic of Uppaal is a subset of the TCTL. In the following we discuss the size of the parse tree of a query in this logic. The size of the parse tree of ϕ, i.e. $|\phi_{parse}|$, equals the number of existential path operators of ϕ located in the nodes of the tree, for which there is a computation of the set of configurations that satisfy ϕ. The truth-values for the basic formulas occurring in the leaves of the parse tree are typically derived from the information of the configurations of the network. Since nested path formulas are not supported by Uppaal, thus, in ϕ occurs one existential path operator.[2] Leaves of the parse tree, i.e. constant true or atomic propositions, do not contribute to the size of ϕ, hence, $|\phi_{parse}| = 1$. Note that $|\phi_{parse}| = |\phi'_{parse}|$ since Ω does not add extra path operators to ϕ'. □

Theorem 5

Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, such that for some $Y \in \mathcal{EC}_{\mathcal{N}}$, some clocks in Y are reset by some complex edges in $CompEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let S be the number of configurations obtained in Theorem 3. Let T be the number of transitions which induce at most S configurations in $\mathcal{N}'$. Let ϕ be a query over $\mathcal{N}$ and $\phi' := \Omega(\phi)$. Let ϕ'_{parse} denote the parse tree of ϕ' and $|\phi'_{parse}|$ its size. Then the time complexity of the TCTL model checking algorithm on the input $\mathcal{N}'$ is bounded from above by

$$(S + T) \cdot |\phi'_{parse}|.$$

Proof. [*Sketch*]. Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, such that for some $Y \in \mathcal{EC}_{\mathcal{N}}$, some clocks in Y are reset by some complex edges in $CompEdges_Y(\mathcal{N})$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements in $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let ϕ be a query over $\mathcal{N}$ and $\phi' := \Omega(\phi)$. Let ϕ'_{parse} denote the parse tree of ϕ' and $|\phi'_{parse}|$ its size.

The factor $|\phi'_{parse}|$ is the same as in the proof sketch for Theorem 4. The first summand, i.e. S, is the bound for the number of configurations of $\mathcal{N}'$ obtained in Theorem 3. The second summand, i.e. T, is a bound for the number of transitions which induce at most S configurations of $\mathcal{N}'$. The bound T considers the number of transitions induced by: (1) delay transitions and, (2) transitions that combine two steps: firstly time passes and then an action transition is taken. Note that in $\mathcal{N}'$ since token t_x-variables related to the same equivalence class are updated by transformed complex edges, and the interleavings of those edges are preserved, thus T considers all possible transitions justified by taking transformed complex edges. Recall that the size of DBMs for the case of $\mathcal{N}'$ is quadratic in the size of $\mathcal{EC}_{\mathcal{N}}$, thus T considers less transitions wrt. operations on zones, as compared to the number of transitions to manipulate zones of $\mathcal{N}$. □

[2] For the special case of a formula using the leads-to operator, the number of existential path operators in that formula is two.

In Chapter 8 we will observe that for the case of a transformed network with transformed complex edges, even when the size of its state-space is bigger than the one of the original network, speedups of the model checking procedure on that transformed network are obtained and justified by a reduced computational effort related to a more efficient DBMs-management.

Chapter 8

Experiments

8.1 Introduction

In this section we present the experimental results that we have obtained when model checking two groups of networks with quasi-equal clocks. The first group consists of nine industrial case studies, for which we model check the original networks and two respective networks transformed with two versions of our Algorithm $\mathcal{K}$. These two versions are, the version presented in this work which implements a broadcast synchronisation for transformed simple edges, and the version from [9] which simulates that broadcast synchronisation with rendez-vous synchronisations.

We obtained our industrial case studies from the scientific literature and projects with small companies in Germany. Each of our industrial case studies is a distributed system that follows a time-triggered architecture.

Predicting savings of configurations in our industrial case studies, is in general a very difficult task due to the involved interleavings occurring in those case studies. Therefore, we create a second group of networks consisting of *toy examples* with reduced interleavings, where predicting savings is easier than with the industrial case studies.

8.2 Source-to-Source Transformation

In Chapter 5 we present transformations for networks of timed automata with quasi-equal clocks and for queries on those networks. Those transformations are implemented in the tool *sAsEt* [12] as fully automatic source-to-source transformations which yield models optimised for verification. In those optimised models further techniques for the analysis of networks can directly be applied, and the modelling language of Uppaal can be as well supported. Thus, with a source-to-source transformation, network designers are able to focus on the design of a network of timed automata, its validation and maintenance. Hence, designers may rely on our transformation for implementing technicalities related to optimisations for verification.

sAsEt is a rich tool built in *Ocaml*. We use sAsEt to automatically detect equivalence classes of quasi-equal clocks, simple and complex edges, and to transform networks of timed automata. We have extended sAsEt by implementing two versions of our approach:

- The *broadcast version*. This version uses Algorithm $\mathcal{K}$ from this work which implements broadcast transitions of transformed simple edges, since the version 4.1.19 of Uppaal allows clock constraints on edges with inputs on broadcast channels.

- The *rendez-vous version*. This version uses the version of Algorithm $\mathcal{K}$ from [9] which simulates broadcast transitions of transformed simple edges as fixed sequences of rendez-vous transitions. This algorithm supports versions of Uppaal (4.0.13 and lower) which do not allow clock constraints on edges with inputs on broadcast channels.

8.3 Preprocessing of Case Studies

For comparison purposes with previous versions of our approach, we implement Algorithm $\mathcal{K}^\nabla$ (cf. Definition 33) on networks transformed with Algorithm $\mathcal{K}$ (Definition 27). The graphical representation of Algorithm $\mathcal{K}^\nabla$ is shown in Figure 8.1.

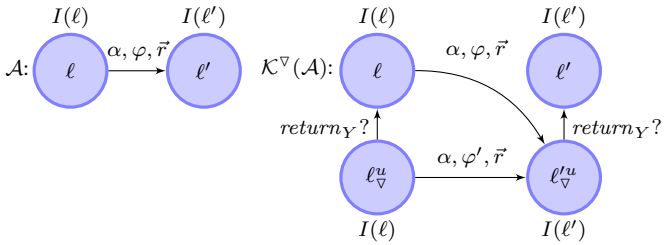


FIGURE 8.1: Transformation pattern of algorithm $\mathcal{K}^\nabla$ over $\mathcal{EC}_\mathcal{N}$ and network $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_\mathcal{N})$, where $\mathcal{A} \in \mathcal{N}'$, and the guard $\varphi' = \varphi \wedge \bigwedge_{Y \in \mathcal{EC}_\mathcal{N}, Y \cap \mathcal{X}(\mathcal{A}) \neq \emptyset} \sum_{x \in Y} t_x > 0$. Algorithm $\mathcal{K}^\nabla$ takes each edge of network $\mathcal{N}'$ (excluding edges of resetters), cf. left-hand side, and transforms it according to the right-hand side. The edge $(\ell, \alpha, \varphi, \vec{r}, \ell')$ originally linking locations ℓ and ℓ' is redirected to ℓ'^u_∇ if and only if $\exists\, Y \in \mathcal{EC}_\mathcal{N}\ \exists\, x \in Y \bullet t_x \in vars(\vec{r}) \wedge t_x \notin V(\mathcal{N})$. Intuitively, only transformed complex and simple edges are redirected.

Definition 33 (Transformation Algorithm $\mathcal{K}^\nabla$)

Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_\mathcal{N} = \{Y_1, \ldots, Y_m\}$. Let $\mathcal{EC}^{prio}_\mathcal{N}$ be a list of all elements of $\mathcal{EC}_\mathcal{N}$, and let $\mathcal{N}' = \{\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{R}_{Y_1}, \ldots, \mathcal{R}_{Y_m}\}$ be the output $\mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_\mathcal{N})$.

The output of $\mathcal{K}^\nabla$ is $\mathcal{N}^\nabla = \{\mathcal{K}^\nabla(\mathcal{A}_1, \mathcal{EC}_\mathcal{N}), \ldots, \mathcal{K}^\nabla(\mathcal{A}_n, \mathcal{EC}_\mathcal{N})\} \cup \{\mathcal{R}_Y \in \mathcal{N}' \mid Y \in \mathcal{EC}_\mathcal{N}\}$, where $\mathcal{K}^\nabla(\mathcal{A}, \mathcal{EC}_\mathcal{N}) = (L', B(\mathcal{A}), \mathcal{X}(\mathcal{A}), V(\mathcal{A}), I', E', \ell_{ini})$ such that:

- $L' = L(\mathcal{A}) \cup \{\ell_\nabla \mid \ell \in L(\mathcal{A})\}$, i.e. one fresh copied location, called ∇-location, for each location in $\mathcal{A}$ is added.
- $I' = \{\ell_\nabla \mapsto I(\ell) \wedge z' \leq 0 \mid \ell \in L(\mathcal{A}) \setminus L^c(\mathcal{A})\}$, i.e. ∇-locations are made urgent by using a fresh clock z'. Note that this clock is not added by this algorithm but internally used by Uppaal in urgent locations.

•

$$E' = E(\mathcal{A}) \setminus \{(\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{A}) \mid \exists Y \in \mathcal{EC}_{\mathcal{N}} \exists\, x \in Y \bullet\, t_x \in vars(\vec{r}) \\ \wedge\, t_x \notin V(\mathcal{N})\} \cup \{(\ell, \alpha, \varphi, \vec{r}, \ell'_{\triangledown}) \mid (\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{A}) \wedge \\ \exists Y \in \mathcal{EC}_{\mathcal{N}}\, \exists\, x \in Y \bullet\, t_x \in vars(\vec{r}) \wedge t_x \notin V(\mathcal{N})\} \cup \\ \{(\ell_{\triangledown}, \alpha, \mho(\varphi), \vec{r}, \ell'_{\triangledown}) \mid (\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{A})\} \\ \cup \{(\ell_{\triangledown}, return_Y?, true, \langle\rangle, \ell), (\ell'_{\triangledown}, return_Y?, true, \langle\rangle, \ell') \mid \\ (\ell, \alpha, \varphi, \vec{r}, \ell') \in E(\mathcal{A})\},$$

i.e. the set of edges E' consists of original edges, redirected edges, copied edges, and edges linking copied and original locations, where $\mho(\varphi) = \varphi \wedge \bigwedge_{Y \in \mathcal{EC}_{\mathcal{N}}, Y \cap \mathcal{X}(\mathcal{A}) \neq \emptyset} \sum_{x \in Y} t_x > 0$. ♢

In a network $\mathcal{N}^{\triangledown} = \mathcal{K}^{\triangledown}(\mathcal{N}', \mathcal{EC}_{\mathcal{N}})$ neither new automata nor new variables nor new clocks are added, but a copy of the structure of each non-resetter automaton existing in the input network $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_{\mathcal{N}})$, where $\mathcal{N}$ is a network with quasi-equal clocks.

In previous versions of our approach this copied structure help us to ensure that time does not elapse as soon as a transformed complex edge is taken, since that edge is redirected to an urgent destination location in the copied structure. Recall that as opposed to transformed simple edges, transformed complex ones do not synchronise with resetters which after taking this synchronisation ensure that time does not elapse by transiting to their respective location $\ell_{nst\mathcal{R}_Y}$.

Note that a carelessly constructed copied structure would double the state-space of transformed networks. However, the copied structure has been carefully constructed and linked by Algorithm $\mathcal{K}^{\triangledown}$ to the original one from $\mathcal{N}'$. That copied structure is transited if and only if the corresponding original one is transited in zero time at time points where representative clocks are reset. Thus, our construction avoids enabling transitions to the original and to the copied structure at the same time. Hence, the size of the reachable state-space of $\mathcal{N}^{\triangledown}$ wrt. $\mathcal{N}'$ is the same.

There exists a standard strong bisimulation between $\mathcal{N}'$ and $\mathcal{N}^{\triangledown}$, and since our case studies are transformed with Algorithm $\mathcal{K}^{\triangledown}$, we have slightly modified Function Ω (Definition 28) to include $\triangledown$-locations in our query transformations. We have as well implemented Algorithm $\mathcal{K}^{\triangledown}$ in sAsEt.

8.4 Industrial Case Studies: Experimental Results

In this section we present and compare the results obtained from model checking networks of the following nine industrial case studies: *FS* [13], *CR* [14], *CD* [15], *LS* [16], *EP* [17], *TT* [18] *TA* [19], *PG* [20] and *FB* [21].

We can group our case studies as implementations of the following time-triggered architectures: (a) **TDMA**: *FS*, *TA* and *CD*, (b) **Pragmatic General Multicast**: *PG*, (c) **Foundation Fieldbus Data Link Layer**: *FB*, (d) **TTEthernet**: *TT*, (e) **Ethernet PowerLink**: *EP*, and (f) **others**: *CR* and *LS*. For more information wrt. the nine industrial case studies we refer the interested reader to the respective sources.

Network	C	kStates	M	$t(s)$	Network	C	kStates	M	$t(s)$	Network	C	kStates	M	$t(s)$
EP-20	21	3,145.7	556.3	485.8	*FS-7*	13	6,532.2	669.4	704.8	*PG-9*	13	152.5	11.7	704.5
EP-20$\mathcal{K}^{\triangledown}$	1	3,145.7	826.5	310.1	*FS-7*$\mathcal{K}^{\triangledown}$	5	3,645.3	5,728.5	121.8	*PG-9*$\mathcal{K}^{\triangledown}$	3	242.8	15.6	5.9
EP-23	24	25,165.8	4,462.5	5,921.0	*FS-9*	15	16,910.2	1,844.4	3,300.7	*PG-11*	15	692.2	29.5	17,763.6
EP-23$\mathcal{K}^{\triangledown}$	1	25,165.9	6,783.9	3,078.9	*FS-9*$\mathcal{K}^{\triangledown}$	5	3,701.3	5,756.8	134.2	*PG-11*$\mathcal{K}^{\triangledown}$	3	1,106.0	45.1	31.8
EP-24	25	–	–	–	*FS-10*	16	–	–	–	*PG-12*	16	–	–	–
EP-24$\mathcal{K}^{\triangledown}$	1	50,331.7	12,506.5	6,640.1	*FS-126*$\mathcal{K}^{\triangledown}$	5	6,982.7	7,580.9	2,102.4	*PG-18*$\mathcal{K}^{\triangledown}$	3	202,113.1	6,234.1	9,179.0
TT-5	6	319.2	25.8	6.7	*CD-12*	13	8,527.7	959.8	453.0					
TT-5$\mathcal{K}^{\triangledown}$	1	327.1	42.5	5.2	*CD-12*$\mathcal{K}^{\triangledown}$	1	98.4	21.7	3.1	Experimental Environment: AMD-Opteron 6174,				
TT-6	7	1,873.0	123.1	36.3	*CD-14*	15	89,396.8	9,492.4	6,457.8	2.2GHz, 16GB, Debian 3.16.39-1,				
TT-6$\mathcal{K}^{\triangledown}$	1	1,916.6	217.0	26.9	*CD-14*$\mathcal{K}^{\triangledown}$	1	458.8	73.4	18.1	verifyta 4.1.19 / default options.				
TT-7	8	10,847.6	625.3	236.3	*CD-15*	16	–	–	–					
TT-7$\mathcal{K}^{\triangledown}$	1	11,054.9	1,232.3	197.2	*CD-15*$\mathcal{K}^{\triangledown}$	1	983.1	146.8	42.9					
LS-6	18	145.1	18.4	6.3	*CR-6*	6	264.5	31.0	3.4					
LS-6$\mathcal{K}^{\triangledown}$	4	166.4	24.3	4.1	*CR-6*$\mathcal{K}^{\triangledown}$	1	181.3	30.7	2.8	sAsEt transformed each of our				
LS-8	22	2,251.3	224.2	150.1	*CR-7*	7	7,223.7	662.0	116.4	benchmarks in at most 2 seconds.				
LS-8$\mathcal{K}^{\triangledown}$	4	2,337.2	284.7	82.4	*CR-7*$\mathcal{K}^{\triangledown}$	1	3,533.7	618.7	67.1					
LS-10	26	33,743.8	3,401.2	3,427.3	*CR-8*	8	97,539.5	11,178.6	2,333.8					
LS-10$\mathcal{K}^{\triangledown}$	4	34,451.5	5,556.2	1,561.8	*CR-8*$\mathcal{K}^{\triangledown}$	1	27,392.4	5,312.9	593.2					
FB-14	16	98.3	25.2	424.7	*TA-2*	7	34.0	9.3	0.4					
FB-14$\mathcal{K}^{\triangledown}$	3	98.3	23.7	6.6	*TA-2*$\mathcal{K}^{\triangledown}$	2	40.1	11.5	0.4	Instances with the mark "–" yielded no results				
FB-16	18	393.2	86.7	9,427.0	*TA-3*	8	795.0	71.3	14.5	in 24 hours.				
FB-16$\mathcal{K}^{\triangledown}$	3	393.2	76.7	33.3	*TA-3*$\mathcal{K}^{\triangledown}$	2	949.5	106.3	13.2					
FB-17	19	–	–	–	*TA-4*	9	25,091.2	1,726.3	608.1					
FB-23$\mathcal{K}^{\triangledown}$	3	50,331.7	8,732.3	8,604.0	*TA-4*$\mathcal{K}^{\triangledown}$	2	31,545.2	2,699.7	579.0					

TABLE 8.1: Row X-$N(\mathcal{K}^{\triangledown})$ gives the figures for benchmark X with N components (and $\mathcal{K}^{\triangledown}$ applied, denoted by the suffix $\mathcal{K}^{\triangledown}$ in the name). 'C' gives the number of clocks in the network, 'kStates' the number of 10^3 states explored, 'M' used memory in MB, and '$t(s)$' verification time in seconds.

We have model checked the following two versions of each case study: (1) the original network and, (2) the network transformed by Algorithm $\mathcal{K}^{\triangledown}$ on the output of Algorithm $\mathcal{K}$ (the broadcast version) from this work.

Table 8.1 gives figures for the verification of *safety queries* in instances of the original and the network transformed with Algorithm $\mathcal{K}^{\triangledown}$ (denoted by the suffix $\mathcal{K}^{\triangledown}$ in the name). This table provides a summary of the verification results obtained for each industrial case study. The interested reader finds in Appendix B an extensive list of instances and their respective verification results obtained for each industrial case study.

In Table 8.1 rows without results indicate the smallest instances of a case study for which we did not obtain results within 24 hours. For that case study we were able to verify queries in at least one more instance of the transformed network than in the original one.

We want to point out that the overhead induced by parsing transformed queries in Uppaal is imperceptible in the verification results of Table 8.1.

8.4.1 Savings of Memory Consumption

The biggest savings of memory consumption are obtained in the transformed networks of *FS*, *CD* and *CR* since those networks have transformed simple edges. As expected by Theorem 2, the size of the state-space of those transformed networks is smaller than the size of the state-space of the respective original networks. For instance, in Table 8.1 we observe savings of at least 44% wrt. the number of states explored in transformed networks of the case study *FS*.

Note in Table 8.1 that verifying queries in transformed networks of the case studies *EP*, *LS*, *PG*, *TT* and *TA* consumes more memory than verifying queries in the original counterparts.

To the memory consumption mainly contributes the size of the state-space of the network being model checked, and the size of the data structures, e.g. DBMs, used in Uppaal to represent zones of that network. The DBMs used to represent zones of transformed networks are more compact than those DBMs for the original networks.

The size of the state-space (in number of configurations) of transformed networks of the case studies *EP*, *LS*, *PG*, *TT* and *TA* is bigger than the size of the state-space of the respective original counterparts (see the theoretical justification in Theorem 3). This is expected since those transformed networks: (1) contain transformed complex edges, (2) preserve all configurations induced by all possible interleavings of those edges, and (3) contain as well configurations induced by interleavings of resetters. Moreover, the mechanism that prioritises transitions in Uppaal contributes as well to the overall memory consumption (we discuss this issue in Example 8). That mechanism is implemented in all networks transformed with Algorithm $\mathcal{K}$ while original networks do not require it.

8.4.2 Savings of Verification Time

In Table 8.1 we show that for all transformed networks except for those of the case study *TA*, we achieve savings of verification time of at least 16.5% wrt. to the verification time in the original networks. For *TA* we achieve savings of at most 9%. The biggest savings of verification time are obtained in the transformed networks of the case studies *FS*, *CD*, *CR*, *FB* and *PG*.

The networks *FS*, *CD*, *CR* contain only simple edges, while the networks *FB* and *PG* contain only complex edges. The reasons for these savings in the cases *FS*, *CD*, *CR* are:

1. In transformed networks of those cases the state-space to be explored when we verify queries, is smaller than the state-space of the original networks. In those transformed networks we encode those configurations which in the original networks are induced by all possible permutations of resets of quasi-equal clocks carried out by simple edges. Hence, there are less configurations to explore (see kStates in Table 8.1).

2. According to [11] the basic operations on DBMs are divided in two groups, namely, *property checking* and *transformation*. The property checking group includes operations to check whether a zone satisfies a given property. The transformation group includes operations to transform zones, e.g. clock reset and time delay. Hence, a compact representation of DBMs (as obtained by using representative clocks in transformed networks) reduces the number of operations on them, and increments the efficiency of DBMs-management.

3. Recall from [42] that during the model checking procedure, Uppaal checks for enabledness of edges which generates an overhead that contributes to the overall verification time. In original networks with simple edges there exists an enabledness check each time that a simple edge is taken. In the respective transformed networks that overhead is reduced since we synchronise enabled transformed simple edges with a broadcast transition.

A theoretical justification for the savings of verification time in the cases *FS*, *CD*, *CR* is provided in Theorem 4.

The quasi-equal clocks in the original networks of *FB*, *PG*, *EP*, *LS*, *TT* and *TA* are reset by complex edges, so all configurations induced by all possible permutations of resets of those clocks are preserved in the respective transformed networks. This explains that the savings in transformed networks of those mentioned case studies are related to a more efficient DBM-management, and not to a reduction of their underlying state-space. A theoretical justification for the savings of verification time in those case studies is provided in Theorem 5.

8.4.3 Algorithm $\mathcal{K}$: Broadcast Versus Rendez-Vous Versions

Regarding savings when comparing the verification results of networks where Algorithm $\mathcal{K}^{\triangledown}$ has been applied on outputs of the broadcast version of Algorithm $\mathcal{K}$ from this work, and on outputs of the rendez-vous version of Algorithm $\mathcal{K}$ from [9], we obtain savings wrt. configurations of at most 4%, and wrt. verification time of at most 9% for networks of the case study *FS* transformed with the broadcast version. Recall that the version of Algorithm $\mathcal{K}$ from this work implements a broadcast synchronisation for transformed simple edges, and the version from [9] simulates that broadcast synchronisation with rendez-vous synchronisations. Thereby, savings are more significant in transformed networks with a large number of transformed simple edges, as in the networks of *FS*.

8.4.4 Predicting Lower Bounds of the Number of Configurations

We predict lower bounds of the number of reachable configurations of transformed networks for the cases *EP* and *FB*. In each of these cases there exists only one equivalence class of quasi-equal clocks, and no other automata but those with complex edges interleave when those clocks are reset.

Regarding *EP*, we know that its set of reset configurations RC contains 2 configurations in each instance. Considering *EP*-20 we obtain the size of its set of reachable configurations, i.e. $|Reach_{\mathcal{N}}|$, from Table B.4 which equals $3,145,794$ configurations. Using the $ComplexEdge-$Formula, i.e. $|Reach_{\mathcal{N}}| + (|RC| \cdot 2)$, from Lemma 5 we predict a lower bound for *EP*-20$\mathcal{K}^{\triangledown}$ as follows, $3,145,794 + (2 \cdot 2) = 3,145,798$, which indeed holds since Table B.4 gives $3,145,799$ states for *EP*-20$\mathcal{K}^{\triangledown}$.

We repeat the same exercise for *FB*, we know that its set of reset configurations RC contains 16 configurations in each instance. Considering *FB*-16 we obtain the size of its set of reachable configurations from Table B.3 which equals $393,243$ configurations. Then we predict a lower bound for *FB*-16$\mathcal{K}^{\triangledown}$ as follows, $393,243 + (16 \cdot 2) = 393,275$, which indeed holds since the same number of states is reported for *FB*-16$\mathcal{K}^{\triangledown}$ in Table B.3.

8.5 Toy Examples: Experimental Results

In this section we present several toys examples whose simplicity allows us to: (1) predict savings in number of configurations, (2) predict lower and upper bounds of the number of reachable configurations of transformed networks and, (3) show costs in memory consumption and verification time of the prioritisation mechanism implemented in transformed networks.
Regarding our toy examples, the broadcast version of Algorithm $\mathcal{K}$ and Algorithm $\mathcal{K}^{\triangledown}$ have been applied in the way already mentioned.

Example 7 (Prediction of Savings in Number of Configurations) *We construct network $\mathcal{N}$ (see Figure 8.2 for a partial view of this network) and use it as toy example. The basic configuration for $\mathcal{N}$ consists of 10 clocks and 10 resetting automata resetting each of those clocks with simple edges. There exist as well 6 non-resetting automata with no clocks dedicated to interleave. We know that the interleavings of the non-resetting automata induce $2,048$ configurations* ***at any time point****.*

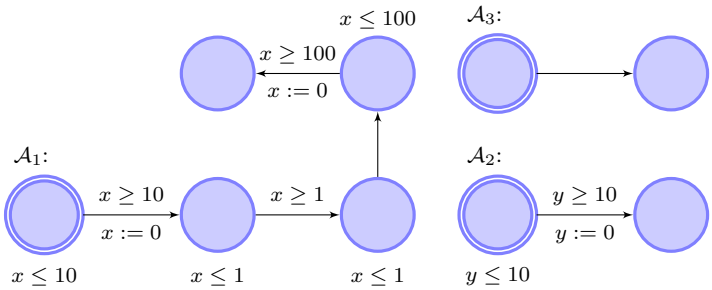


FIGURE 8.2: Examples of automata of network $\mathcal{N}$ used in Example 7. Automata of $\mathcal{N}$ can be grouped in three groups, where only the fist two groups use quasi-equal clocks. Group 1: automata syntactically similar to $\mathcal{A}_1$. Group 2: automata syntactically similar to $\mathcal{A}_2$. Group 3: automata with no clocks dedicated to interleave, where $\mathcal{A}_3$ is an example.

There are 5 *equivalence classes, i.e.* A, B, C, D *and* E*, with two clocks each. In our experiments we instantiate only clocks of the equivalence class* A*, while the other classes remain with the same number of clocks.*

We know that there are 3 *reset configurations, i.e.* $|RC| = 3$*. In the first reset configuration the clocks of classes* A *and* B *are ready to be reset; in the second the clocks of* C, D *and* E*, and in the third only the clocks of* A*.*

The results of verifying a safety property expressing that time is never stopped either at origin or at destination locations of simple edges, are given in Table 8.2. Then we use the $SimpleEdge-Formula$ *from Lemma 3, i.e.* $|Reach_{\mathcal{N}}| - \left(\sum_{s \in RC} 2^{|clks(s)|}\right) + \sum_{s \in RC} \Big[|class(s)| + 2\Big]$*, to calculate the number of configurations of the transformed instance, say* $\mathcal{N}$*-1*$\mathcal{K}^{\triangledown}$*. From Table 8.2 the number of configurations of* $\mathcal{N}$*-1, i.e. the size of* $Reach_{\mathcal{N}}$*, equals* $184,320$*. Using the formula from above yields,*

$$184,320 - ((2,048 \cdot 2^4) + (2,048 \cdot 2^6) + (2,048 \cdot 2^2)) +$$
$$(2,048 \cdot (2+2)) + (2,048 \cdot (3+2)) + (2,048 \cdot (1+2)) = 36,864.$$

Considering that our implementation of Algorithm $\mathcal{K}$ *induces one extra configuration for initialisation of variables, the number* $36,864 + 1$ *equals the number of states reported in Table 8.2 for* $\mathcal{N}$*-1*$\mathcal{K}^{\triangledown}$*. Hence, for* $\mathcal{N}$*-1*$\mathcal{K}^{\triangledown}$ *we save* $147,455$ *configurations by synchronising transformed simple edges with broadcast transitions.*

The same exercise for the instance, say $\mathcal{N}$*-7, where* $|A| = 8$*, yields,*

$$16,183,296 - ((2,048 \cdot 2^{10}) + (2,048 \cdot 2^6) + (2,048 \cdot 2^8)) +$$
$$(2,048 \cdot (2+2)) + (2,048 \cdot (3+2)) + (2,048 \cdot (1+2)) = 13,455,360,$$

which by adding the extra configuration for initialisation of variables equals the number of states reported for $\mathcal{N}$*-7*$\mathcal{K}^{\triangledown}$*. Hence, for* $\mathcal{N}$*-7*$\mathcal{K}^{\triangledown}$ *we save* $2,727,936$ *configurations by synchronising transformed simple edges with broadcast transitions.*

$\diamondsuit$

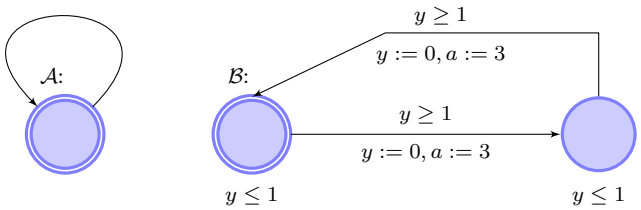


FIGURE 8.3: Network P used in Example 8.

Network	C	States	M	$t(s)$	Network	C	States	M	$t(s)$
$\mathcal{N}$-1	10	184,320	30.8	7.84	$\mathcal{N}$-1$\mathcal{K}^{\nabla}$	5	36,865	12.4	1.5
$\mathcal{N}$-2	11	262,144	44.0	13.4	$\mathcal{N}$-2$\mathcal{K}^{\nabla}$	5	73,729	18.6	3.4
$\mathcal{N}$-3	12	454,656	66.0	28.8	$\mathcal{N}$-3$\mathcal{K}^{\nabla}$	5	184,321	36.7	9.5
$\mathcal{N}$-4	13	950,272	131.9	80.8	$\mathcal{N}$-4$\mathcal{K}^{\nabla}$	5	516,097	90.6	29.6
$\mathcal{N}$-5	14	2,273,280	328.5	246.0	$\mathcal{N}$-5$\mathcal{K}^{\nabla}$	5	1,511,425	259.4	99.3
$\mathcal{N}$-6	15	5,914,624	817.7	787.0	$\mathcal{N}$-6$\mathcal{K}^{\nabla}$	5	4,497,409	783.2	330.0
$\mathcal{N}$-7	16	16,183,296	2,157.4	2,696.2	$\mathcal{N}$-7$\mathcal{K}^{\nabla}$	5	13,455,361	2,140.8	1,096.9

TABLE 8.2: Results for the toy example $\mathcal{N}$, where all clocks are reset by simple edges.

Example 8 (The Cost of Using Priorities) *This toy example is the network P (see Figure 8.3) whose basic configuration consists of two automata: A with no clocks and no variables, a single location and a self-loop, and B with a variable and a quasi-equal clock which belongs to the equivalence class Y. Both clock and variable are reset by the same non-looped complex edge with no synchronisation channel.*

In our experiments we instantiate only automaton B, so that the size of Y increases. There exist two versions of transformed networks, namely, P' and P'' which preserve all interleavings justified by the original complex edges, and which only differ on the operation of the mechanism for prioritising transitions. That is, in version P' the mechanism for prioritising transitions is activated, while in version P'' is not.

This toy example is specially constructed to isolate and observe the effects on the memory consumption and verification time of the mechanism for prioritising transitions. Although not reported, the number of reachable configurations of P' and P'' is the same, hence, prioritising transitions in this case has no effect in the overall number of reachable configurations. Obviously, the difference wrt. the number of reachable configurations between an instance of P and any its corresponding transformed version, is the number of configurations induced by the transitions of the resetter in the transformed network.

In Figure 8.4 we report the results of memory consumption and verification time of a safety property in each network. DBMs in P' and P'' are more compact than in P this contributes to less memory consumption and less verification time of properties in transformed networks. Moreover, Figure 8.4 clearly shows that the implementation of the mechanism for prioritising transitions in P' increases the overall memory consumption and verification time wrt. P'', where that mechanism is not operative.

Without looking into the code of Uppaal we can only expeculate how this tool internally prioritises transitions. One idea is that in Uppaal the prioritisation is not statically implemented but dynamically, so that Uppaal internally stores an ordering of transitions in a data structure and refreshes it each time that, for instance, a successor configuration is computed. ♢

Example 9 (Predicting Upper Bounds of the Number of Configurations) *We construct network $\mathcal{N}$ (consisting of automata syntactically similar to automaton $\mathcal{A}_2$ of Figure 8.2) and use it as toy example. The basic configuration for $\mathcal{N}$ consists of 6 clocks and 6 automata, each with a unique edge which is a non-looped complex edge with no synchronisation channel and that resets each clock.*

There are 3 equivalence classes, i.e. A, B and C, with two clocks each. In our experiments we instantiate only clocks of the equivalence class A, while the other classes remain with the same number of clocks.

We know that there is only 1 reset configuration, i.e. $|RC| = 1$, where all clocks are ready to be reset. Considering, say $\mathcal{N}$-1, we predict an upper bound of the number of configurations of its respective transformed network. To this end, from Table 8.3 we obtain the size of its set of reachable configurations, i.e. $|Reach_{\mathcal{N}}|$, which equals 64 configurations.

The upper bound $ComplexEdge{-}Formula$, i.e. $|Reach_{\mathcal{N}}| \cdot \big(\sum_{s \in RC} 2^{|class(s)|}\big)$, from Lemma 6 yields an upper bound for $\mathcal{N}$-1$\mathcal{K}^{\triangledown}$ of, $(64) \cdot ((1) \cdot 2^3) = 512$ configurations. The upper bound holds since Table 8.3 reports 201 states for $\mathcal{N}$-1$\mathcal{K}^{\triangledown}$.

The same exercise for, say $\mathcal{N}$-18, predicts an upper bound of $(8,388,608) \cdot ((1) \cdot 2^3) = 67,108,864$ configurations of the respective transformed network. The upper bound holds since Table 8.3 reports $18,350,141$ states for $\mathcal{N}$-18$\mathcal{K}^{\triangledown}$. ♢

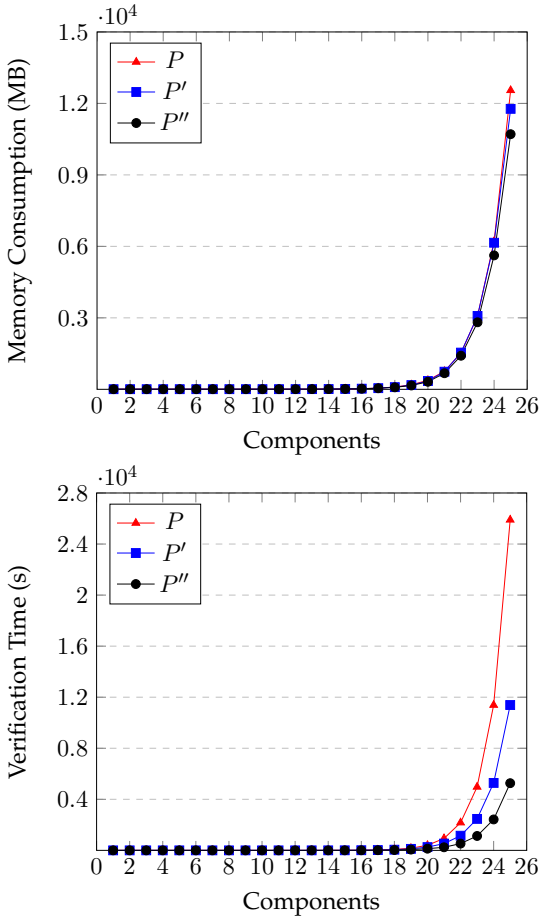


FIGURE 8.4: Verification results for the toy example P and its two transformed networks.

Network	C	States	M	$t(s)$	Network	C	States	M	$t(s)$
$\mathcal{N}$-*1*	6	64	5.7	0.0	$\mathcal{N}$-*1*$\mathcal{K}^{\triangledown}$	3	201	6.1	0.0
$\mathcal{N}$-*2*	7	128	5.8	0.0	$\mathcal{N}$-*2*$\mathcal{K}^{\triangledown}$	3	341	6.3	0.0
$\mathcal{N}$-*3*	8	256	5.9	0.0	$\mathcal{N}$-*3*$\mathcal{K}^{\triangledown}$	3	621	6.4	0.0
$\mathcal{N}$-*4*	9	512	6.0	0.0	$\mathcal{N}$-*4*$\mathcal{K}^{\triangledown}$	3	1,181	6.5	0.0
$\mathcal{N}$-*5*	10	1,024	6.1	0.0	$\mathcal{N}$-*5*$\mathcal{K}^{\triangledown}$	3	2,301	6.7	0.0
$\mathcal{N}$-*6*	11	2,048	6.3	0.0	$\mathcal{N}$-*6*$\mathcal{K}^{\triangledown}$	3	4,541	7.2	0.1
$\mathcal{N}$-*7*	12	4,096	6.6	0.2	$\mathcal{N}$-*7*$\mathcal{K}^{\triangledown}$	3	9,021	8.0	0.3
$\mathcal{N}$-*8*	13	8,192	7.5	0.5	$\mathcal{N}$-*8*$\mathcal{K}^{\triangledown}$	3	17,981	9.5	0.7
$\mathcal{N}$-*9*	14	16,384	8.9	1.4	$\mathcal{N}$-*9*$\mathcal{K}^{\triangledown}$	3	35,901	12.5	1.7
$\mathcal{N}$-*10*	15	32,768	11.8	3.5	$\mathcal{N}$-*10*$\mathcal{K}^{\triangledown}$	3	71,741	18.9	3.9
$\mathcal{N}$-*11*	16	65,536	20.2	8.5	$\mathcal{N}$-*11*$\mathcal{K}^{\triangledown}$	3	143,421	31.6	8.8
$\mathcal{N}$-*12*	17	131,072	35.2	20.4	$\mathcal{N}$-*12*$\mathcal{K}^{\triangledown}$	3	286,781	56.4	20.4
$\mathcal{N}$-*13*	18	262,144	65.9	49.1	$\mathcal{N}$-*13*$\mathcal{K}^{\triangledown}$	3	573,501	106.8	45.4
$\mathcal{N}$-*14*	19	524,288	127.8	115.8	$\mathcal{N}$-*14*$\mathcal{K}^{\triangledown}$	3	1,146,941	206.5	103.5
$\mathcal{N}$-*15*	20	1,048,576	252.0	274.7	$\mathcal{N}$-*15*$\mathcal{K}^{\triangledown}$	3	2,293,821	405.3	227.2
$\mathcal{N}$-*16*	21	2,097,152	502.3	639.6	$\mathcal{N}$-*16*$\mathcal{K}^{\triangledown}$	3	4,587,581	836.7	516.5
$\mathcal{N}$-*17*	22	4,194,304	1,006.3	1,492.3	$\mathcal{N}$-*17*$\mathcal{K}^{\triangledown}$	3	9,175,101	1,666.0	1,114.3
$\mathcal{N}$-*18*	23	8,388,608	2,023.1	3,435.0	$\mathcal{N}$-*18*$\mathcal{K}^{\triangledown}$	3	18,350,141	3,324.0	2,409.7

TABLE 8.3: Results for the toy example $\mathcal{N}$ where all clocks are reset by complex edges.

Chapter 9

Conclusion

In this thesis, we present a fully automatic reduction of the number of quasi-equal clocks in any network of timed automata. Our approach yields transformed networks where all properties of the original networks are reflected. The cost of verifying properties is much lower in transformed networks than in their original versions with quasi-equal clocks. Our approach does not impose any syntactical assumption neither on networks nor on single automata, thus we are able to transform **the whole class of timed automata with quasi-equal clocks**.

Considering a network with a set of equivalence classes of quasi-equal clocks, where all clocks are exclusively reset by simple edges. In this thesis we demonstrate that the worst-case space complexity of the model checking algorithm on the transformed version of that network, is exponential in the number of equivalence classes of quasi-equal clocks, despite the additional modelling elements added to the transformed version.

We present a source-to-source transformation which is automatically delivered by our tool *sAsEt*. Moreover, sAsEt implements distinct versions of our transformation in order to support several versions of Uppaal. This source-to-source transformation outputs a network of timed automata where: (1) further techniques for the analysis of networks can directly be applied, (2) the full query language of Uppaal is supported and, (3) the rich modelling language of Uppaal can be supported.

We provide formulas to quantify the number of reachable configurations in transformed automata, and thereby know the number of configurations saved in those automata. For certain transformed automata our quantification is exact, and for the other automata we bound that number of reachable configurations.

We apply our quasi-equal clocks reduction approach on nine industrial cases studies, and provided extensive experimental results which show significant savings in verification costs.

We compare the savings in memory consumption and verification time delivered by the distinct versions of our transformation, and we predict the exact size of some transformed versions of our industrial cases studies.

Future Work. We plan to extend the results obtained in this thesis, e.g. source-to-source transformation, formulas to quantify reachable configurations, etc., to the quasi-dependent variables reduction approach [46]. Currently, that approach imposes strong assumptions (similar to those in [7]) on networks of hybrid automata. With the knowledge gained from the quasi-equal clocks reduction approach we are able to eliminate those assumptions. Preliminary work in this direction has been carried out in [54].

Appendix A

Weak Bisimulation and Auxiliary Propositions

A.1 Introduction

We recommend to begin with Lemma 2 (page 105), and then read these functions and propositions as required by the lemma.

Definition 34 (Functions $\delta_{s,r}$, ϖ and ζ)
Let $\mathcal{N}$ be a network with $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}})$ Let $s \in \mathit{Conf}(\mathcal{N})$ and $r \in \mathit{Conf}(\mathcal{N}')$, such that $r \in \mathit{QE}(s)$. Then:

1. $\delta_{s,r} : \mathit{LogVars} \to \{0,1\}$ is a function defined point-wise as follows,

$$\delta_{s,r}(\tilde{x}) := \begin{cases} 1 & \text{, if } \exists\, 1 \le i \le n, 1 \le j \le m, \\ & (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in \mathit{SimpEdges}_{Y_j}(\mathcal{A}_i) \bullet\ \ell_{s,i} = \ell \wedge \\ & \ell_{r,i} = \ell' \wedge \ell_{r,\mathcal{R}_{Y_j}} = \ell_{nst\mathcal{R}_{Y_j}}\ \wedge\ \nu_s(x) = \nu_r(\mathit{rep}(x)) \wedge \\ & \nu_r(t_x) = 0, \\ 0 & \text{, otherwise.} \end{cases}$$

2. $\varpi : (\mathit{Conf}(\mathcal{N}) \times \mathit{Conf}(\mathcal{N}') \times \mathbb{N} \times (\mathit{LogVars} \to \{0,1\})) \to L(\mathcal{N})$ is a function defined point-wise as follows, $\varpi(s, r, i, \delta) :=$

$$\begin{cases} \ell & \text{, if } \exists\, \ell', \alpha, \varphi, x, Y \bullet (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_i) \wedge \\ & \big((\delta(\tilde{x}) = 1\ \wedge \ell_{r,i} = \ell') \vee (\ell_{r,i} = \ell)\big), \\ \ell' & \text{, if } \exists\, \ell, \alpha, \varphi, x, Y \bullet (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_i) \wedge \\ & \delta(\tilde{x}) = 0\ \wedge \ell_{r,i} = \ell', \\ \ell_{s,i} & \text{, otherwise.} \end{cases}$$

3. $\zeta : (\mathit{Conf}(\mathcal{N}) \times \mathit{Conf}(\mathcal{N}') \times \mathcal{X}(\mathcal{N}) \times (\mathit{LogVars} \to \{0,1\})) \to \mathit{Time}$ is a function defined point-wise as follows, $\zeta(s, r, x, \delta) :=$

$$\begin{cases} \nu_r(\mathit{rep}(x)) & \text{, if } \exists\, i, \ell, \alpha, \varphi, \ell', Y \bullet (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_i) \\ & \quad \wedge \big((\delta(\tilde{x}) = 1\ \wedge \ell_{r,i} = \ell' \wedge \nu_r(t_x) = 0) \vee \\ & \quad (\ell_{r,i} = \ell \wedge \nu_r(t_x) = 1)\big), \\ 0 & \text{, if } \exists\, i, \ell, \alpha, \varphi, \ell', Y \bullet (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_i) \\ & \quad \wedge\ \delta(\tilde{x}) = 0\ \wedge \ell_{r,i} = \ell' \wedge \nu_r(t_x) = 0, \\ \nu_s(x) & \text{, otherwise.} \end{cases}$$

$\diamondsuit$

Proposition 1

Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}})$. Let $s \in \mathit{Conf}(\mathcal{N})$ be a configuration of $\mathcal{N}$. Let $\kappa_{\mathcal{N}}$ be as defined in Definition 29. Then

$$\forall\, r \in \mathit{QE}(s) \bullet r \models_{\delta_{s,r}} \kappa_{\mathcal{N}}.$$

Proof. Let $\mathcal{N} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}} = \{Y_1, \ldots, Y_m\}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}})$. Let $s \in \mathit{Conf}(\mathcal{N})$ be a configuration of $\mathcal{N}$.

Pick $r \in \mathit{QE}(s)$. By Function Ω, $\kappa_{\mathcal{N}}$ consists of the following conjunction of implications:

$$\bigwedge_{\substack{1 \le i \le n,\\ 1 \le j \le m,\\ (\ell,\alpha,\varphi,\langle x:=0\rangle,\ell') \in \mathit{SimpEdges}_{Y_j}(\mathcal{A}_i)}} \kappa(x).$$

We claim that each implication $\kappa(x)$ holds as follows:

- $\kappa(x) =$

$$(\tilde{x} \implies \bigvee_{(\ell,\alpha,\varphi,\langle x:=0\rangle,\ell') \in \mathit{SimpEdges}_{Y_j}(\mathcal{A}_i)} \ell' \wedge \ell_{\mathit{nst}\mathcal{R}_{Y_j}}).$$

  We distinguish the following values that Function $\delta_{s,r}$ assigns to $\tilde{x}$.

  - $\delta_{s,r}(\tilde{x}) = 0$. Then the implication $\kappa(x)$ is trivially true and satisfied by r.
  - $\delta_{s,r}(\tilde{x}) = 1$. The premise $\tilde{x} = 1$ of $\kappa(x)$ is true and we now carry out a consistency check for the value assigned to $\tilde{x}$, by checking that the conclusion of $\kappa(x)$ is also true. Since $\delta_{s,r}(\tilde{x}) = 1$, then by Definition 34 of Function $\delta_{s,r}$ there exists a simple edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in \mathit{SimpEdges}_{Y_j}(\mathcal{A}_i)$, such that the ith automaton in r is located at ℓ', i.e. $\ell_{r,i} = \ell'$, and the resetter automaton $\mathcal{R}_{Y_j}$ in r is located at $\ell_{\mathit{nst}\mathcal{R}_{Y_j}}$, i.e. $\ell_{r,\mathcal{R}_{Y_j}} = \ell_{\mathit{nst}\mathcal{R}_{Y_j}}$. From the conclusion of $\kappa(x)$ (which is a disjunction of clauses), the configuration r satisfies the disjunct $(\ell' \wedge \ell_{\mathit{nst}\mathcal{R}_{Y_j}})$. Hence, r satisfies the conclusion of $\kappa(x)$. Thus, the implication $\kappa(x)$ is satisfied by r.

□

Proposition 2

Let $\mathcal{N} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}} = \{Y_1, \ldots, Y_m\}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}})$. Let $s \in \mathit{Conf}(\mathcal{N})$ and $r \in \mathit{Conf}(\mathcal{N}')$ be two configurations such that $r \in \mathit{QE}(s)$. Let CF be a configuration formula over $\mathcal{N}$. Then

A. $s \models_\delta \mathit{CF} \implies r \models_\delta \Omega(\mathit{CF})$.

B. $r \models_\delta \Omega(\mathit{CF}) \implies \exists\, \dot{s} \in \mathit{Conf}(\mathcal{N}) \bullet r \in \mathit{QE}(\dot{s}) \wedge \dot{s} \models_\delta \mathit{CF}$.

Proof. Let $p = |\mathcal{X}(\mathcal{N})|$. Let CV be the set of logical variables introduced by Ω in transformations for locations and clock constraints occurring in CF.

- By induction over the structure of CF we prove the following claim

$$s \models_\delta CF \implies r \models_{\delta[\tilde{x}_i := \delta_{s,r}(\tilde{x}_i) | 1 \leq i \leq p, \tilde{x}_i \in CV]} \Omega_0(CF), \qquad (\star)$$

which implies the claim A as follows:

$s \models_\delta CF$
$\implies$ By $(\star)$.
$r \models_{\delta[\tilde{x}_i := \delta_{s,r}(\tilde{x}_i) | 1 \leq i \leq p, \tilde{x}_i \in CV]} \Omega_0(CF)$
$\implies$ By Proposition 1.
$r \models_{\delta[\tilde{x}_i := \delta_{s,r}(\tilde{x}_i) | 1 \leq i \leq p, x_i \in CV]} \Omega_0(CF) \wedge \kappa_{\mathcal{N}}$
$\implies$ Set $d_i := \delta_{s,r}(\tilde{x}_i)$.
$\exists\, d_1, \ldots, d_p \in \{0,1\} \bullet$
$r \models_{\delta[\tilde{x}_i := d_i | 1 \leq i \leq p, \tilde{x}_i \in CV]} \Omega_0(CF) \wedge \kappa_{\mathcal{N}}$
$\iff$ By Definition 18.
$r \models_\delta \exists\, \tilde{x}_1, \ldots, \tilde{x}_p \bullet \Omega_0(CF) \wedge \kappa_{\mathcal{N}}$
$\iff$ By Function Ω.
$r \models_\delta \Omega(CF)$

In the following we show $(\star)$. Let $\delta' := \delta[\tilde{x}_i := \delta_{s,r}(\tilde{x}_i) \mid 1 \leq i \leq p, \tilde{x}_i \in CV]$.

  - Base case i. $CF = \varphi_{int}$. We show that $s \models_\delta \varphi_{int} \iff r \models_{\delta'} \Omega_0(\varphi_{int})$.

    $s \models_\delta \varphi_{int}$
    $\iff$ By Definition 18.
    $\nu_s \models_\delta \varphi_{int}$
    $\iff$ Since φ_{int} is a constraint over $V(\mathcal{N})$.
    $\nu_s|_{V(\mathcal{N})} \models_\delta \varphi_{int}$
    $\iff$ By $(\dagger)$.
    $\nu_r|_{V(\mathcal{N})} \models_{\delta'} \varphi_{int}$
    $\iff$ Since $\Omega_0(\varphi_{int}) = \varphi_{int}$.
    $\nu_r|_{V(\mathcal{N})} \models_{\delta'} \Omega_0(\varphi_{int})$
    $\iff$ Since φ_{int} is a constraint over $V(\mathcal{N})$.
    $\nu_r \models_{\delta'} \Omega_0(\varphi_{int})$
    $\iff$ By Definition 18.
    $r \models_{\delta'} \Omega_0(\varphi_{int})$

    $(\dagger)$ By Rule (6.2.1): $\nu_s(x) = \nu_r(x)$, for each $x \in V(\mathcal{N})$. Moreover, variables occurring in δ' are fresh ones and do not occur in φ_{int}.

  - Base case ii. Let $\mathcal{A}_j$, with $1 \leq j \leq n$, be an automaton of $\mathcal{N}$, $CF = \mathcal{A}_j.\ell$, such that ℓ is neither an origin nor a destination location of a simple edge. We show that $s \models_\delta \mathcal{A}_j.\ell \iff r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell)$.

$s \models_\delta \mathcal{A}_j.\ell$
$\iff$ By $s = \langle \vec{\ell}_s, \nu_s \rangle, t$.
$\langle \vec{\ell}_s, \nu_s \rangle, t \models_\delta \mathcal{A}_j.\ell$
$\iff$ By Definition 18.
$\ell_{s,j} = \mathcal{A}_j.\ell$
$\iff$ By Rule (6.0.2a): $\ell_{s,j} = \ell_{r,j}$.
$\ell_{r,j} = \mathcal{A}_j.\ell$
$\iff$ By Definition 18 and $\Omega_0(\mathcal{A}_j.\ell) = \mathcal{A}_j.\ell$.
$\langle \vec{\ell}_r, \nu_r \rangle, t \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell)$
$\iff$ By $r = \langle \vec{\ell}_r, \nu_r \rangle, t$.
$r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell)$

Note that Rule (6.0.2b) does not apply in this base case since in that rule $\mathcal{A}_j$ is located at the origin location of a simple edge.

– Base case iii. Let $\mathcal{A}_j$, with $1 \leq j \leq n$, be an automaton of $\mathcal{N}$, $CF = \mathcal{A}_j.\ell$, where ℓ is the origin of the simple edge $(\ell, \alpha, \varphi, \langle x := 0 \rangle, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_j)$, for some $Y \in \mathcal{EC}_\mathcal{N}$. We show that $s \models_\delta \mathcal{A}_j.\ell \implies r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell)$.

$s \models_\delta \mathcal{A}_j.\ell$
$\iff$ By $s = \langle \vec{\ell}_s, \nu_s \rangle, t$.
$\langle \vec{\ell}_s, \nu_s \rangle, t \models_\delta \mathcal{A}_j.\ell$
$\iff$ By Definition 18.
$\ell_{s,j} = \mathcal{A}_j.\ell$
$\implies$ By (†).
$\ell_{r,j} = \mathcal{A}_j.\ell \vee$
$(\ell_{r,j} = \mathcal{A}_j.\ell' \wedge\ \delta_{s,r}(\tilde{x}) = 1)$
$\iff$ By Definition 18 and
$\Omega_0(\mathcal{A}_j.\ell) = (\mathcal{A}_j.\ell \vee (\mathcal{A}_j.\ell' \wedge \tilde{x}))$.
$\langle \vec{\ell}_r, \nu_r \rangle, t \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell)$
$\iff$ By $r = \langle \vec{\ell}_r, \nu_r \rangle, t$.
$r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell)$

(†) Since $r \in \mathit{QE}(s)$ then either Rule (6.0.2a) or Rule (6.0.2b) holds for this base case. We make the following distinctions.

* If Rule (6.0.2a) holds then the jth automaton in both configurations is located at the same location, i.e. $\ell_{s,j} = \ell_{r,j}$, thus, $\ell_{r,j} = \mathcal{A}_j.\ell$ holds.
* If Rule (6.0.2b) holds then the jth automaton in both configurations, and automaton $\mathcal{R}_Y$ are located as follows, $\ell_{s,j} = \mathcal{A}_j.\ell, \ell_{r,j} = \mathcal{A}_j.\ell'$, $\ell_{r,\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y}$ and $\nu_r(t_x) = 0$, therefore, 1 is assigned to $\tilde{x}$ by Function $\delta_{s,r}$ (Definition 34), thus, $(\ell_{r,j} = \mathcal{A}_j.\ell' \wedge\ \delta_{s,r}(\tilde{x}) = 1)$ holds.

– Base case iv. Let $\mathcal{A}_j$, with $1 \leq j \leq n$, be an automaton of $\mathcal{N}$, $CF = \mathcal{A}_j.\ell'$, such that ℓ' is the destination of the simple edge $(\ell, \alpha, \varphi, \langle x := 0 \rangle, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_j)$, for some $Y \in \mathcal{EC}_\mathcal{N}$. We show that $s \models_\delta \mathcal{A}_j.\ell' \implies r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell')$.

$s \models_\delta \mathcal{A}_j.\ell'$
$\iff$ By $s = \langle \vec{\ell}_s, \nu_s \rangle, t$.
$\langle \vec{\ell}_s, \nu_s \rangle, t \models_\delta \mathcal{A}_j.\ell'$
$\iff$ By Definition 18.
$\ell_{s,j} = \mathcal{A}_j.\ell'$
$\implies$ By (†).
$\ell_{r,j} = \mathcal{A}_j.\ell' \wedge \neg(\delta_{s,r}(\tilde{x}) = 0)$
$\iff$ By Definition 18 and $\Omega_0(\mathcal{A}_j.\ell') = (\mathcal{A}_j.\ell' \wedge \neg\tilde{x})$.
$\langle \vec{\ell}_r, \nu_r \rangle, t \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell')$
$\iff$ By $r = \langle \vec{\ell}_r, \nu_r \rangle, t$.
$r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell')$

(†) By Rule (6.0.2a) the jth automaton in both configurations is located at the same location, i.e. $\ell_{s,j} = \ell_{r,j}$. However, $\ell_{s,j} \neq \ell$, hence, Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$.

Note that Rule (6.0.2b) does not apply in this base case since in that rule $\mathcal{A}_j$ is located at the origin location of a simple edge.

– Base case v. Let $\mathcal{A}_j$, with $1 \leq j \leq n$, be an automaton of $\mathcal{N}$. Let $Y \in \mathcal{EC}_\mathcal{N}$. Let $e = (\ell, \alpha, \varphi, \vec{r}, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_j)$ be a simple edge. Let $CF = x \sim c$, with $x \in \mathcal{X}(\mathcal{A}_j)$. We show that $s \models_\delta x \sim c \implies r \models_{\delta'} \Omega_0(x \sim c)$.

$s \models_\delta x \sim c$
$\iff$ By $s = \langle \vec{\ell}_s, \nu_s \rangle, t$.
$\nu_s \models_\delta x \sim c$
$\iff$ By Definition 18.
$\nu_s(x) \sim c$
$\implies$ By (†).
$(\nu_r(\mathit{rep}(x)) \sim c \wedge$
$(\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1)) \vee$
$(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge$
$\neg(\delta_{s,r}(\tilde{x}) = 1))$
$\iff$ By Definition 18.
$\nu_r \models_{\delta'} (\mathit{rep}(x) \sim c \wedge (t_x \vee \tilde{x}))$
$\vee (0 \sim c \wedge \neg t_x \wedge \neg\tilde{x})$
$\iff$ By Definition 18 and Ω_0 over $x \sim c$.
$\langle \vec{\ell}_r, \nu_r \rangle, t \models_{\delta'} \Omega_0(x \sim c)$
$\iff$ By $r = \langle \vec{\ell}_r, \nu_r \rangle, t$.
$r \models_{\delta'} \Omega_0(x \sim c)$

(†) Since $r \in \mathit{QE}(s)$ then either Rule (6.0.2a) or Rule (6.0.2b) holds for this base case. We make the following distinctions.

1. If Rule (6.0.2a) holds then the jth automaton in both configurations is located at the same location, i.e. $\ell_{s,j} = \ell_{r,j}$, and either (a) $\nu_s(x) = \nu_s(\mathit{rep}(x))$ or (b) $\nu_s(x) \neq \nu_s(\mathit{rep}(x))$. We distinguish:
   (a) $\nu_s(x) = \nu_s(\mathit{rep}(x))$. By Rule (6.0.2a), $\nu_s(x) = \nu_r(\mathit{rep}(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 1$. Because of the value of t_x in r, the conjunction $(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1))$

yields false. Because $\ell_{s,j} = \ell_{r,j}$, Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$, and from $(\nu_r(rep(x)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1))$ the conjunction $(\nu_r(rep(x)) \sim c \ \wedge \nu_r(t_x) = 1)$ holds.

(b) $\nu_s(x) \neq \nu_s(rep(x))$. By Rule (6.0.2a), $\nu_s(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 0$. Because $\ell_{s,j} = \ell_{r,j}$, Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$, and since $\nu_r(t_x) = 0$, then $(\nu_r(rep(x)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1))$ yields false. Because of the mentioned values for t_x and $\tilde{x}$ the conjunction $(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1))$ holds.

2. If Rule (6.0.2b) holds then $\ell_{s,j} = \ell$, $\ell_{r,j} = \ell'$ and $\ell_{r,\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y}$. We distinguish the following cases:

(a) x is reset by e. By Rule (6.0.2b), $\nu_s(x) = \nu_r(rep(x))$ and $\nu_r(t_x) = 0$. Function $\delta_{s,r}$ assigns 1 to $\tilde{x}$, since $\ell_{s,j} = \ell$, $\ell_{r,j} = \ell'$, $\ell_{r,\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y}$ and $\nu_r(t_x) = 0$. Because of the value of $\tilde{x}$, the conjunction $(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1))$ yields false. Since $\nu_r(t_x) = 0$, from $(\nu_r(rep(x)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1))$ the conjunction $(\nu_r(rep(x)) \sim c \ \wedge \delta_{s,r}(\tilde{x}) = 1)$ holds.

(b) x is not reset by e. By Rule (6.0.2b), $\nu_s(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 1$. Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$, since $\nu_r(t_x) = 1$. Because of the value of t_x, the conjunction $(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1))$ yields false. Since $\tilde{x}$ has value 0, from $(\nu_r(rep(x)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1))$ the conjunction $(\nu_r(rep(x)) \sim c \wedge \nu_r(t_x) = 1)$ holds.

– Base case vi. Let $\mathcal{A}_j$ and $\mathcal{A}_k$, with $1 \leq j \leq k \leq n$, be automata of $\mathcal{N}$. Let $e_1 = (\ell_j, \alpha_j, \varphi_j, \vec{r}_j, \ell'_j) \in \mathit{SimpEdges}_Y(\mathcal{A}_j)$ and $e_2 = (\ell_k, \alpha_k, \varphi_k, \vec{r}_k, \ell'_k) \in \mathit{SimpEdges}_W(\mathcal{A}_k)$, with $Y, W \in \mathcal{EC}_\mathcal{N}$, be simple edges. Let $CF = x - y \sim c$, with $x \in \mathcal{X}(\mathcal{A}_j)$ and $y \in \mathcal{X}(\mathcal{A}_k)$. We show that $s \models_\delta x - y \sim c \implies r \models_{\delta'} \Omega_0(x - y \sim c)$.

$$
\begin{aligned}
& s \models_\delta x - y \sim c \\
& \iff && \text{By } s = \langle \vec{\ell}_s, \nu_s \rangle, t. \\
& \nu_s \models_\delta x - y \sim c \\
& \iff && \text{By Definition 18.} \\
& \nu_s(x) - \nu_s(y) \sim c \\
& \implies && \text{By } (\dagger). \\
& \Big(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \wedge \\
& (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \ \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \vee \\
& \Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \\
& (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \vee \\
& \Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \\
& \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big) \vee \\
& \Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \\
& \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big) \\
& \iff && \text{By Definition 18.} \\
& \nu_r \models_{\delta'} \Big(rep(x) - rep(y) \sim c \ \wedge (t_x \vee \tilde{x}) \ \wedge (t_y \vee \tilde{y})\Big) \vee \\
& \Big(0 - rep(y) \sim c \ \wedge \neg t_x \wedge \neg \tilde{x} \ \wedge (t_y \vee \tilde{y})\Big) \vee \\
& \Big(rep(x) - 0 \sim c \ \wedge (t_x \vee \tilde{x}) \ \wedge \neg t_y \wedge \neg \tilde{y}\Big) \vee \\
& \Big(0 \sim c \ \wedge \neg t_x \wedge \neg \tilde{x} \ \wedge \neg t_y \wedge \neg \tilde{y}\Big) \\
& \iff && \text{By Definition 18.} \\
& && \text{and } \Omega_0 \text{ over} \\
& && x - y \sim c. \\
& \langle \vec{\ell}_r, \nu_r \rangle, t \models_{\delta'} \Omega_0(x - y \sim c) \\
& \iff && \text{By } r = \langle \vec{\ell}_r, \nu_r \rangle, t. \\
& r \models_{\delta'} \Omega_0(x - y \sim c)
\end{aligned}
$$

($\dagger$) Since $r \in QE(s)$ then either Rule (6.0.2a) or Rule (6.0.2b) holds for x and y. We carry out the following distinctions.

1. If Rule (6.0.2a) holds for x then $\ell_{s,j} = \ell_{r,j}$ and either $\nu_s(x) = \nu_s(rep(x))$ or $\nu_s(x) \neq \nu_s(rep(x))$. If Rule (6.0.2a) holds for y then $\ell_{s,k} = \ell_{r,k}$ and either $\nu_s(y) = \nu_s(rep(y))$ or $\nu_s(y) \neq \nu_s(rep(y))$. We distinguish the following cases:
   (a) $\nu_s(x) = \nu_s(rep(x))$ and $\nu_s(y) = \nu_s(rep(y))$. By Rule (6.0.2a) for x, $\nu_s(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 1$. By Rule (6.0.2a) for y, $\nu_s(y) = \nu_r(rep(y)) \cdot \nu_r(t_y)$, where $\nu_r(t_y) = 1$. Because of the values of t_x and t_y in r, the following disjunction yields false:

$$\Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big)$$
$$\vee\Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big)$$
$$\vee\Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big).$$

Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$ and to $\tilde{y}$, since $\ell_{s,j} = \ell_{r,j}$, $\ell_{s,k} = \ell_{r,k}$, respectively. Therefore, from $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \ (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)$ the conjunction $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge \nu_r(t_x) = 1 \wedge \nu_r(t_y) = 1)$ holds.

(b) $\nu_s(x) \neq \nu_s(rep(x))$ and $\nu_s(y) = \nu_s(rep(y))$. By Rule (6.0.2a) for x, $\nu_s(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 0$. By Rule (6.0.2a) for y, $\nu_s(y) = \nu_r(rep(y)) \cdot \nu_r(t_y)$, where $\nu_r(t_y) = 1$. Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$ and to $\tilde{y}$, since $\ell_{s,j} = \ell_{r,j}$, $\ell_{s,k} = \ell_{r,k}$, respectively. Because of the mentioned values of t_x, t_y, $\tilde{x}$ and $\tilde{y}$ the following disjunction yields false:

$$\Big(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big)$$
$$\vee\Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big)$$
$$\vee\Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

and from $(0 - \nu_r(rep(y)) \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \ \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)$ the conjunction $(0 - \nu_r(rep(y)) \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \nu_r(t_y) = 1)$ holds.

(c) $\nu_s(x) = \nu_s(rep(x))$ and $\nu_s(y) \neq \nu_s(rep(y))$. This case is symmetrical to the previous one.

(d) $\nu_s(x) \neq \nu_s(rep(x))$ and $\nu_s(y) \neq \nu_s(rep(y))$. By Rule (6.0.2a) for x, $\nu_s(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 0$. By Rule (6.0.2a) for y, $\nu_s(y) = \nu_r(rep(y)) \cdot \nu_r(t_y)$, where $\nu_r(t_y) = 0$. Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$ and to $\tilde{y}$, since $\ell_{s,j} = \ell_{r,j}$, $\ell_{s,k} = \ell_{r,k}$, respectively. By the values of t_x, t_y,

$\tilde{x}$ and $\tilde{y}$ the following disjunction yields false:

$$\Big(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \\ \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \\ \vee\Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \\ (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \\ \vee\Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \\ \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

and the conjunction $(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \ \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1))$ holds.

2. If Rule (6.0.2a) holds for x then $\ell_{s,j} = \ell_{r,j}$ and either $\nu_s(x) = \nu_s(rep(x))$ or $\nu_s(x) \neq \nu_s(rep(x))$. If Rule (6.0.2b) holds for y then e_2 may reset y. We distinguish the following cases:
   (a) $\nu_s(x) = \nu_s(rep(x))$ and e_2 resets y. By Rule (6.0.2a) for x, $\nu_s(x) = \nu_r(rep(x))\cdot\nu_r(t_x)$, where $\nu_r(t_x) = 1$. Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$ since $\ell_{s,j} = \ell_{r,j}$. By Rule (6.0.2b) and since e_2 resets y, hence, $\ell_{s,k} = \ell_k$, $\ell_{r,k} = \ell'_k$, $\ell_{r,\mathcal{R}_W} = \ell_{nst\mathcal{R}_W}$ and $\nu_r(t_y) = 0$, therefore, Function $\delta_{s,r}$ assigns 1 to $\tilde{y}$. By the values of t_x, t_y, $\tilde{x}$ and $\tilde{y}$ the following disjunction yields false:

$$\Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \\ (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \\ \vee\Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \\ \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big) \\ \vee\Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \\ \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

   and from $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \ (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)$ the conjunction $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge \nu_r(t_x) = 1 \wedge \delta_{s,r}(\tilde{y}) = 1)$ holds.
   (b) $\nu_s(x) = \nu_s(rep(x))$ and e_2 does not reset y. By Rule (6.0.2a) for x, $\nu_s(x) = \nu_r(rep(x))\cdot\nu_r(t_x)$, where $\nu_r(t_x) = 1$. Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$ since $\ell_{s,j} = \ell_{r,j}$. By Rule (6.0.2b) and since e_2 does not reset y, hence, $\nu_s(y) = \nu_r(rep(y)) \cdot \nu_r(t_y)$, where $\nu_r(t_y) = 1$, therefore, Function $\delta_{s,r}$ assigns 0 to $\tilde{y}$. By the values of t_x, t_y, $\tilde{x}$ and $\tilde{y}$ the

following disjunction yields false:

$$\Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \vee \Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big) \vee \Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

and from $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \ (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)$ the conjunction $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge \nu_r(t_x) = 1 \wedge \nu_r(t_y) = 1)$ holds.

(c) $\nu_s(x) \neq \nu_s(rep(x))$ and e_2 resets y. By Rule (6.0.2a) for x, $\nu_s(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 0$. Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$ since $\ell_{s,j} = \ell_{r,j}$. By Rule (6.0.2b) and since e_2 resets y, hence, $\ell_{s,k} = \ell_k$, $\ell_{r,k} = \ell'_k$, $\ell_{r,\mathcal{R}_W} = \ell_{nst\mathcal{R}_W}$ and $\nu_r(t_y) = 0$, therefore, Function $\delta_{s,r}$ assigns 1 to $\tilde{y}$. By the values of t_x, t_y, $\tilde{x}$ and $\tilde{y}$ the following disjunction yields false:

$$\Big(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \vee \Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big) \vee \Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

and from $(0 - \nu_r(rep(y)) \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \ \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1))$ the conjunction $(0 - \nu_r(rep(y)) \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \delta_{s,r}(\tilde{y}) = 1)$ holds.

(d) $\nu_s(x) \neq \nu_s(rep(x))$ and e_2 does not reset y. By Rule (6.0.2a) for x, $\nu_s(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 0$. Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$ since $\ell_{s,j} = \ell_{r,j}$. By Rule (6.0.2b) and since e_2 does not reset y, hence, $\nu_s(y) = \nu_r(rep(y)) \cdot \nu_r(t_y)$, where $\nu_r(t_y) = 1$, therefore, Function $\delta_{s,r}$ assigns 0 to $\tilde{y}$. By the values of t_x, t_y, $\tilde{x}$ and $\tilde{y}$ the

following disjunction yields false:

$$\Big(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \vee \Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big) \vee \Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

and from $(0 - \nu_r(rep(y)) \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \ \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)$ the conjunction $(0 - \nu_r(rep(y)) \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \delta_{s,r}(\tilde{y}) = 1)$ holds.

3. If Rule (6.0.2b) holds for x then $\ell_{s,j} = \ell_j$, $\ell_{r,j} = \ell'_j$, $\ell_{r,\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y}$ and e_1 may reset x. If Rule (6.0.2a) holds for y then $\ell_{s,k} = \ell_{r,k}$ and either $\nu_s(y) = \nu_s(rep(y))$ or $\nu_s(y) \neq \nu_s(rep(y))$. This case is symmetrical to the previous one.
4. If Rule (6.0.2b) holds for x then $\ell_{s,j} = \ell_j$, $\ell_{r,j} = \ell'_j$, $\ell_{r,\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y}$ and e_1 may reset x. If Rule (6.0.2b) holds for y then $\ell_{s,k} = \ell_k$, $\ell_{r,k} = \ell'_k$, $\ell_{r,\mathcal{R}_W} = \ell_{nst\mathcal{R}_W}$ and e_2 may reset y.
   (a) e_1 resets x and e_2 resets y. By Rule (6.0.2b) and since e_1 resets x, hence, $\ell_{s,j} = \ell_j$, $\ell_{r,j} = \ell'_j$, $\ell_{r,\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y}$ and $\nu_r(t_x) = 0$, therefore, Function $\delta_{s,r}$ assigns 1 to $\tilde{x}$. By Rule (6.0.2b) and since e_2 resets y, hence, $\ell_{s,k} = \ell_k$, $\ell_{r,k} = \ell'_k$, $\ell_{r,\mathcal{R}_W} = \ell_{nst\mathcal{R}_W}$ and $\nu_r(t_y) = 0$, therefore, Function $\delta_{s,r}$ assigns 1 to $\tilde{y}$. By the values of t_x, t_y, $\tilde{x}$ and $\tilde{y}$ the following disjunction yields false:

$$\Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big) \vee \Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big) \vee \Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

and from $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \ (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1))$ the conjunction $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge \delta_{s,r}(\tilde{x}) = 1 \wedge \delta_{s,r}(\tilde{y}) = 1)$ holds.

   (b) e_1 resets x and e_2 does not reset y. By Rule (6.0.2b) and since e_1 resets x, hence, $\ell_{s,j} = \ell_j$, $\ell_{r,j} = \ell'_j$, $\ell_{r,\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y}$ and $\nu_r(t_x) = 0$, therefore, Function $\delta_{s,r}$ assigns 1 to $\tilde{x}$. By Rule (6.0.2b) and since e_2 does not reset y, hence, $\nu_s(y) =$

$\nu_r(rep(y)) \cdot \nu_r(t_y)$, where $\nu_r(t_y) = 1$, therefore, Function $\delta_{s,r}$ assigns 0 to $\tilde{y}$. By the values of t_x, t_y, $\tilde{x}$ and $\tilde{y}$ the following disjunction yields false:

$$\Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big)$$
$$\vee\Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big)$$
$$\vee\Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

and from $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \ (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1))$ the conjunction $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge \delta_{s,r}(\tilde{x}) = 1 \wedge \nu_r(t_y) = 1)$ holds.

(c) e_1 does not reset x and e_2 resets y. This case is symmetrical to the previous one.

(d) e_1 does not reset x and e_2 does not reset y. By Rule (6.0.2b) and since e_1 does not reset x, hence, $\nu_s(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$, where $\nu_r(t_x) = 1$, therefore, Function $\delta_{s,r}$ assigns 0 to $\tilde{x}$. By Rule (6.0.2b) and since e_2 does not reset y, hence, $\nu_s(y) = \nu_r(rep(y)) \cdot \nu_r(t_y)$, where $\nu_r(t_y) = 1$, therefore, Function $\delta_{s,r}$ assigns 0 to $\tilde{y}$. By the values of t_x, t_y, $\tilde{x}$ and $\tilde{y}$ the following disjunction yields false:

$$\Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1)\Big)$$
$$\vee\Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big)$$
$$\vee\Big(0 \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta_{s,r}(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta_{s,r}(\tilde{y}) = 1)\Big),$$

and from $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta_{s,r}(\tilde{x}) = 1) \wedge \ (\nu_r(t_y) = 1 \vee \delta_{s,r}(\tilde{y}) = 1))$ the conjunction $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge \nu_r(t_x) = 1 \wedge \nu_r(t_y) = 1)$ holds.

– Induction step i. $CF = \neg CF_1$. We show that $s \models_\delta \neg CF_1 \implies r \models_{\delta'} \Omega_0(\neg CF_1)$.

$s \models_\delta \neg CF_1$
$\iff$ By Definition 18.
$\neg(s \models_\delta CF_1)$
$\implies$ By induction assumption.
$\neg(r \models_{\delta'} \Omega_0(CF_1))$
$\iff$ By Definition 18 and Ω_0.
$r \models_{\delta'} \Omega_0(\neg CF_1)$

– Induction step ii. $CF = CF_1 \wedge CF_2$. We show that $s \models_\delta CF_1 \wedge CF_2 \implies r \models_{\delta'} \Omega_0(CF_1 \wedge CF_2)$.
$s \models_\delta CF_1 \wedge CF_2$
$\iff$ By Definition 18.
$s \models_\delta CF_1 \wedge s \models_\delta CF_2$
$\implies$ By induction assumption.
$r \models_{\delta'} \Omega_0(CF_1) \wedge r \models_{\delta'} \Omega_0(CF_2)$
$\iff$ By Definition 18.
$r \models_{\delta'} \Omega_0(CF_1) \wedge \Omega_0(CF_2)$
$\iff$ By Ω_0 Definition 28.
$r \models_{\delta'} \Omega_0(CF_1 \wedge CF_2)$

– Induction step iii. $CF = \exists\, \theta_1, \ldots, \theta_n \bullet CF$. We show that $s \models_\delta \exists\, \theta_1, \ldots, \theta_n \bullet CF \implies r \models_{\delta'} \Omega_0(\exists\, \theta_1, \ldots, \theta_n \bullet CF)$.
$s \models_\delta \exists\, \theta_1, \ldots, \theta_n \bullet CF$
$\iff$ By Definition 18.
$\exists\, \bar{d}_1, \ldots, \bar{d}_m \in \{0,1\} \bullet$
$s \models_{\delta[\theta_i := \bar{d}_i \mid 1 \leq i \leq m]} CF$
$\implies$ By induction assumption.
$\exists\, \bar{d}_1, \ldots, \bar{d}_m \in \{0,1\} \bullet$
$r \models_{\delta'[\theta_i := \bar{d}_i \mid 1 \leq i \leq m]} \Omega_0(CF)$
$\iff$ By Definition 18.
$r \models_{\delta'} \exists\, \theta_1, \ldots, \theta_n \bullet \Omega_0(CF)$
$\iff$ By Ω_0 Definition 28.
$r \models_{\delta'} \Omega_0(\exists\, \theta_1, \ldots, \theta_n \bullet CF)$

- By induction over the structure of CF we prove the following claim:

$$r \models_{\delta'} \Omega_0(CF) \implies r \in QE(\dot{s}_{r,\delta'}) \wedge \dot{s}_{r,\delta'} \models_\delta CF, \qquad (\star\star)$$

which implies the claim B as follows:

$r \models_\delta \Omega(CF)$
$\iff$ By Function Ω.
$r \models_\delta \exists \tilde{x}_1, \ldots, \tilde{x}_p \bullet \Omega_0(CF) \wedge \kappa_\mathcal{N}$
$\iff$ By Definition 18.
$\exists\, d_1, \ldots, d_p \in \{0,1\} \bullet r \models_{\delta[\tilde{x}_i := d_i | 1 \leq i \leq p, x_i \in CV]} \Omega_0(CF) \wedge \kappa_\mathcal{N}$

$\implies$ Let $\delta' := \delta[\tilde{x}_i := d_i \mid 1 \leq i \leq p, x_i \in CV]$, s. t. $r \models_{\delta'} \Omega_0(CF) \wedge \kappa_\mathcal{N}$.
$r \models_{\delta'} \Omega_0(CF) \wedge \kappa_\mathcal{N}$
$\implies$ By Proposition 1.
$r \models_{\delta'} \Omega_0(CF)$
$\implies$ By $(\star\star)$.
$r \in QE(\dot{s}_{r,\delta'}) \wedge \dot{s}_{r,\delta'} \models_\delta CF$
$\implies$ Set $\dot{s} := \dot{s}_{r,\delta'}$.
$\exists\, \dot{s} \in Conf(\mathcal{N}) \bullet r \in QE(\dot{s}) \wedge \dot{s} \models_\delta CF$

In the following we show $(\star\star)$. We construct a configuration $\dot{s}_{r,\delta'}$, such that $r \in QE(\dot{s}_{r,\delta'}) \wedge \dot{s}_{r,\delta'} \models_\delta CF$. Propose configuration $\dot{s}_{r,\delta'} = \langle(\varpi(s,r,1,\delta'), \ldots, \varpi(s,r,n,\delta')), \{x \mapsto \zeta(s,r,x,\delta') \mid x \in \mathcal{X}(\mathcal{N}) \cup V(\mathcal{N})\}\rangle$. In the following we show that $r \in QE(\dot{s}_{r,\delta'})$.

- The value of each variable $v \in V(\mathcal{N})$ is the same in configurations $\dot{s}_{r,\delta'}$ and r, i.e. $\nu_{\dot{s}_{r,\delta'}}(v) = \nu_r(v)$ since ζ does not change values of v. Therefore, Rule (6.2.1) holds.
- Now we discuss the (possible) changes carried out by ϖ. For $1 \leq i \leq n$ and for each $1 \leq j \leq m$, either Rule (6.0.2a) or Rule (6.0.2b) holds:
  - If Rule (6.0.2a) holds then the ith automaton is located at the same location in $\dot{s}_{r,\delta'}$ and r, i.e. $\ell_{\dot{s}_{r,\delta'},i} = \ell_{r,i}$ and the value of each clock $x \in \mathcal{X}(\mathcal{A}_i)$ is related as follows, $\nu_{\dot{s}_{r,\delta'}}(x) = \nu_r(rep(x)) \cdot \nu_r(t_x)$.
  - If Rule (6.0.2b) holds then there exists a simple edge $e = (\ell, \alpha, \varphi, \vec{r}, \ell') \in SimpEdges_{Y_j}(\mathcal{A}_i)$ such that the ith automaton in $\dot{s}_{r,\delta'}$ is located at ℓ, and that automaton in r is located at ℓ', and if e resets $x \in \mathcal{X}(\mathcal{A}_i)$ then the value of x in $\dot{s}_{r,\delta'}$ is the same as the value of the representative $rep(x)$ in r, i.e. $\nu_{\dot{s}_{r,\delta'}}(x) = \nu_r(rep(x))$, and since the ith automaton in r is located at ℓ', thus, $\nu_r(t_x) = 0$. Otherwise, e does not reset x, hence, the value of x is related in both configurations in the following way, $\nu_{\dot{s}_{r,\delta'}}(y) = \nu_r(rep(y)) \cdot \nu_r(t_y)$.
- Rule (6.2.3) holds since the value of the variable $s_Y^{\mathcal{A}_i}$ is consistent with the location of the transformed version of $\mathcal{A}_i$ in r, i.e. $\nu_r(s_Y^{\mathcal{A}_i}) = 1$ if there exists $(\ell, \alpha, \varphi, \vec{r}, \ell') \in SimpEdges_{Y_j}(\mathcal{A}_i)$ such that $\ell_{r,i} = \ell$ and 0 otherwise.
- Rule (6.2.4) holds since the value of the variable $prio_Y$ is consistent with the location of the resetter $\mathcal{R}_Y$ in r, i.e. $\nu_r(prio_Y) = 1$ if $\ell_{r,\mathcal{R}_Y}$ and 0 otherwise.

We now show that $r \models_{\delta'} \Omega_0(CF) \implies \dot{s}_{r,\delta'} \models_\delta CF$. We begin with base case iii, since base cases i ($CF = \varphi_{int}$) and ii ($CF = \ell$, such that ℓ

is neither an origin nor a destination location of a simple edge) have been shown in A.

- Base case iii. Let $\mathcal{A}_j$, with $1 \leq j \leq n$, be an automaton of $\mathcal{N}$, $CF = \mathcal{A}_j.\ell$, such that ℓ is the origin of the edge $e = (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_j)$, for some equivalence class $Y \in \mathcal{EC}_\mathcal{N}$. We show that $r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell) \implies \dot{s}_{r,\delta'} \models_{\delta'} \mathcal{A}_j.\ell$.
  Note that $\Omega_0(\mathcal{A}_j.\ell) = (\mathcal{A}_j.\ell \vee (\mathcal{A}_j.\ell' \wedge \tilde{x}))$ and that

$$r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell) \iff (\ell_{r,j} = \ell \vee (\ell_{r,j} = \ell' \wedge \delta'(\tilde{x}) = 1)). \quad (\dagger)$$

  We make the following distinctions based on the values for the logical variable $\tilde{x}$.
  - $\delta'(\tilde{x}) = 0$. Clearly the clause $(\ell_{r,j} = \ell' \wedge \delta'(\tilde{x}) = 1)$ from the disjunction in $(\dagger)$ yields false. Hence, the clause $(\ell_{r,j} = \ell)$ yields true. By construction of $\dot{s}_{r,\delta'}$, $\mathcal{A}_j$ is located at ℓ since $\varpi(\dot{s}_{r,\delta'}, r, j, \delta')$ outputs $\mathcal{A}_j.\ell$ because $\ell_{r,j} = \ell$. Thus, $\dot{s}_{r,\delta'} \models \mathcal{A}_j.\ell$.
  - $\delta'(\tilde{x}) = 1$. We make the following distinctions:
    - $\ell_{r,j} = \ell \wedge \neg(\ell_{r,j} = \ell')$. Clearly the clause $(\ell_{r,j} = \ell)$ from the disjunction in $(\dagger)$ yields true (the right-hand side clause yields false). By construction of $\dot{s}_{r,\delta'}$, $\mathcal{A}_j$ is located at ℓ, since $\varpi(\dot{s}_{r,\delta'}, r, j, \delta')$ outputs $\mathcal{A}_j.\ell$ because $\ell_{r,j} = \ell$. Thus, $\dot{s}_{r,\delta'} \models \mathcal{A}_j.\ell$.
    - $\neg(\ell_{r,j} = \ell) \wedge \ell_{r,j} = \ell'$. Clearly the clause $(\ell_{r,j} = \ell' \wedge \delta'(\tilde{x}) = 1)$ from the disjunction in $(\dagger)$ yields true (the left-hand side clause yields false). By construction of $\dot{s}_{r,\delta'}$, $\mathcal{A}_j$ is located at ℓ, since $\varpi(\dot{s}_{r,\delta'}, r, j, \delta')$ outputs $\mathcal{A}_j.\ell$ because $\delta'(\tilde{x}) = 1$ and $\ell_{r,j} = \ell'$. Thus, $\dot{s}_{r,\delta'} \models \mathcal{A}_j.\ell$.
    - $\ell_{r,j} = \ell \wedge \ell_{r,j} = \ell'$. Clearly e is a self-looped edge. Either clause from $(\dagger)$ yields true. By construction of $\dot{s}_{r,\delta'}$, $\mathcal{A}_j$ is located at ℓ, since $\varpi(\dot{s}_{r,\delta'}, r, j, \delta')$ outputs $\mathcal{A}_j.\ell$ because $\ell_{r,j} = \ell$, or because $\delta'(\tilde{x}) = 1$ and $\ell_{r,j} = \ell'$. Thus, $\dot{s}_{r,\delta'} \models \mathcal{A}_j.\ell$.
- Base case iv. Let $\mathcal{A}_j$, with $1 \leq j \leq n$, be an automaton of $\mathcal{N}$, $CF = \mathcal{A}_j.\ell'$, such that ℓ' is the destination of the edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in \mathit{SimpEdges}_Y(\mathcal{A}_j)$, for some equivalence class $Y \in \mathcal{EC}_\mathcal{N}$. We show that $r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell') \implies \dot{s}_{r,\delta'} \models_{\delta'} \mathcal{A}_j.\ell'$.
  Note that $\Omega_0(\mathcal{A}_j.\ell') = (\mathcal{A}_j.\ell' \wedge \neg\tilde{x})$ and that $r \models_{\delta'} \Omega_0(\mathcal{A}_j.\ell') \iff (\ell_{r,j} = \ell' \wedge \neg(\delta'(\tilde{x}) = 1))$. Clearly there exists only one value, namely 0, for the logical variable $\tilde{x}$ such that $r \models_{\bar{\delta}} \Omega_0(\mathcal{A}_j.\ell')$.
  By construction of $\dot{s}_{r,\delta'}$, $\mathcal{A}_j$ is located at ℓ', since $\varpi(\dot{s}_{r,\delta'}, r, j, \delta')$ outputs $\mathcal{A}_j.\ell'$ because $\delta'(\tilde{x}) = 0$ and $\ell_{r,j} = \ell'$. Thus, $\dot{s}_{r,\delta'} \models \mathcal{A}_j.\ell'$.
- Base case v. Let $\mathcal{A}_j$, with $1 \leq j \leq n$, be an automaton of $\mathcal{N}$. Let $Y \in \mathcal{EC}_\mathcal{N}$. Let $CF = x \sim c$, with $x \in \mathcal{X}(\mathcal{A}_j)$. We show that $r \models_{\delta'} \Omega_0(x \sim c) \implies \dot{s}_{r,\delta'} \models_{\delta'} (x \sim c)$. Note that $\Omega_0(x \sim c) =$

$(rep(x) \sim c \ \wedge (t_x \vee \tilde{x})) \ \vee (0 \sim c \wedge \neg t_x \wedge \neg \tilde{x})$ and that

$$r \models_{\delta'} \Omega_0(x \sim c) \iff$$
$$(\nu_r(rep(x)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta'(\tilde{x}) = 1)) \vee$$
$$(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \ \wedge \neg(\delta'(\tilde{x}) = 1)).$$

We make the following distinctions based on the values for the logical variable $\tilde{x}$.

* $\delta'(\tilde{x}) = 0$. We distinguish the following cases:
  - $\nu_r(rep(x)) \sim c \wedge (\nu_r(t_x) = 1 \vee \delta'(\tilde{x}) = 1))$. By construction of $\dot{s}_{r,\delta'}$, x obtains from $\zeta(\dot{s}_{r,\delta'}, r, x, \delta')$ either:
    1. $\nu_r(rep(x))$, if there exists an edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_j)$ for some $Y \in \mathcal{EC}_\mathcal{N}$, such that the transformed version of $\mathcal{A}_j$ is located at ℓ in $\dot{s}_{r,\delta'}$ and $\nu_r(t_x) = 1$ or,
    2. $\nu_s(x)$, if the transformed version of $\mathcal{A}_j$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting x.

    Thus, $\dot{s}_{r,\delta'} \models x \sim c$.
  - $(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \ \wedge \neg(\delta'(\tilde{x}) = 1))$. By construction of $\dot{s}_{r,\delta'}$, x obtains from $\zeta(\dot{s}_{r,\delta'}, r, x, \delta')$ either:
    1. 0, if there exists a simple edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_j)$ for some $Y \in \mathcal{EC}_\mathcal{N}$, such that the transformed version of $\mathcal{A}_j$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_x) = 0$ and $\delta'(\tilde{x}) = 0$ or,
    2. $\nu_s(x)$ if the transformed version of $\mathcal{A}_j$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting x.

    Thus, $\dot{s}_{r,\delta'} \models x \sim c$.
* $\delta'(\tilde{x}) = 1$. Clearly the clause $(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \ \wedge \neg(\delta'(\tilde{x}) = 1))$ yields false, hence, $(\nu_r(rep(x)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta'(\tilde{x}) = 1))$ yields true. By construction of $\dot{s}_{r,\delta'}$, x obtains from $\zeta(\dot{s}_{r,\delta'}, r, x, \delta')$ either:
  1. $\nu_r(rep(x))$, if there exists an edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_j)$ for some $Y \in \mathcal{EC}_\mathcal{N}$, such that the transformed version of $\mathcal{A}_j$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_x) = 0$ and $\delta'(\tilde{x}) = 1$ or, that automaton is located at ℓ in $\dot{s}_{r,\delta'}$ and $\nu_r(t_x) = 1$.

  Thus, $\dot{s}_{r,\delta'} \models x \sim c$.

– Base case vi. Let $\mathcal{A}_j$ and $\mathcal{A}_k$, with $1 \leq j \leq k \leq n$, be automata of $\mathcal{N}$. Let $CF = x - y \sim c$, with $x \in \mathcal{X}(\mathcal{A}_j)$ and $y \in \mathcal{X}(\mathcal{A}_k)$. We show that $r \models_{\delta'} \Omega_0(x - y \sim c) \implies \dot{s}_{r,\delta'} \models_{\delta'} (x - y \sim c)$. Note that $\Omega_0(x - y \sim c) =$

$$\big(\big(rep(x) - rep(y) \sim c \wedge \ (t_x \vee \tilde{x}) \ \wedge (t_y \vee \tilde{y})\big) \vee$$
$$\big(0 - rep(y) \sim c \ \wedge \neg t_x \wedge \neg \tilde{x} \ \wedge (t_y \vee \tilde{y})\big) \vee$$
$$\big(rep(x) - 0 \sim c \ \wedge (t_x \vee \tilde{x}) \ \wedge \neg t_y \wedge \neg \tilde{y}\big) \vee$$
$$\big(0 \sim c \ \wedge \neg t_x \wedge \neg \tilde{x} \ \wedge \neg t_y \wedge \neg \tilde{y}\big)\big),$$

and that

$$
\begin{aligned}
r \models_{\delta'} \Omega_0(x - y \sim c) \iff \\
\Big(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta'(\tilde{x}) = 1) \wedge \\
(\nu_r(t_y) = 1 \vee \delta'(\tilde{y}) = 1)\Big) \vee \\
\Big(0 - \nu_r(rep(y)) \sim c \ \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta'(\tilde{x}) = 1) \wedge \\
(\nu_r(t_y) = 1 \vee \delta'(\tilde{y}) = 1)\Big) \vee \\
\Big(\nu_r(rep(x)) - 0 \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta'(\tilde{x}) = 1) \wedge \\
\neg(\nu_r(t_y) = 1) \wedge \neg(\delta'(\tilde{y}) = 1)\Big) \vee \\
\Big(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \ \wedge \neg(\delta'(\tilde{x}) = 1) \wedge \\
\neg(\nu_r(t_y) = 1) \ \wedge \neg(\delta'(\tilde{y}) = 1)\Big). (\dagger)
\end{aligned}
$$

We make the following distinctions based on the values for the logical variables $\tilde{x}$ and $\tilde{y}$.

* $\delta'(\tilde{x}) = 0$ and $\delta'(\tilde{y}) = 0$. We distinguish the following cases:
  - $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \ \wedge (\nu_r(t_x) = 1 \vee \delta'(\tilde{x}) = 1) \ \wedge (\nu_r(t_y) = 1 \vee \delta'(\tilde{y}) = 1))$. By construction of $\dot{s}_{r,\delta'}$, x and y respectively obtain from $\zeta(\dot{s}_{r,\delta'}, r, x, \delta')$ and $\zeta(\dot{s}_{r,\delta'}, r, y, \delta')$ the following values. Firstly x:
    1. $\nu_r(rep(x))$, if there exists an edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_j)$ for some $Y \in \mathcal{EC}_{\mathcal{N}}$, such that the transformed version of $\mathcal{A}_j$ is located at ℓ in $\dot{s}_{r,\delta'}$ and $\nu_r(t_x) = 1$ or,
    2. $\nu_s(x)$, if the transformed version of $\mathcal{A}_j$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting x.

    Secondly y:
    1. $\nu_r(rep(y))$, if there exists an edge $(\ell, \alpha, \varphi, \langle y := 0\rangle, \ell') \in SimpEdges_W(\mathcal{A}_k)$ for some $W \in \mathcal{EC}_{\mathcal{N}}$, such that the transformed version of $\mathcal{A}_k$ is located at ℓ in $\dot{s}_{r,\delta'}$ and $\nu_r(t_y) = 1$ or,
    2. $\nu_s(y)$, if the transformed version of $\mathcal{A}_k$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting y.

    Thus, $\dot{s}_{r,\delta'} \models x - y \sim c$.
  - $(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \ \wedge \neg(\delta'(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \ \wedge \neg(\delta'(\tilde{y}) = 1))$. By construction of $\dot{s}_{r,\delta'}$, x and y respectively obtain from $\zeta(\dot{s}_{r,\delta'}, r, x, \delta')$ and $\zeta(\dot{s}_{r,\delta'}, r, y, \delta')$ the following values. Firstly x:
    1. 0, if there exists a simple edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_j)$ for some $Y \in \mathcal{EC}_{\mathcal{N}}$, such that the transformed version of $\mathcal{A}_j$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_x) = 0$ and $\delta'(\tilde{x}) = 0$ or,

2. $\nu_s(x)$ if the transformed version of $\mathcal{A}_j$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting x.

Secondly y:

1. 0, if there exists a simple edge $(\ell, \alpha, \varphi, \langle y := 0\rangle, \ell') \in SimpEdges_W(\mathcal{A}_k)$ for some $W \in \mathcal{EC}_{\mathcal{N}}$, such that the transformed version of $\mathcal{A}_k$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_y) = 0$ and $\delta'(\tilde{y}) = 0$ or,
2. $\nu_s(y)$ if the transformed version of $\mathcal{A}_k$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting y.

Thus, $\dot{s}_{r,\delta'} \models x - y \sim c$.

$*$ $\delta'(\tilde{x}) = 0$ and $\delta'(\tilde{y}) = 1$. Clearly the following clauses from (†) yield false: $\Big(\nu_r(rep(x)) - 0 \sim c \wedge (\nu_r(t_x) = 1 \vee \delta'(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta'(\tilde{y}) = 1)\Big) \vee \Big(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta'(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta'(\tilde{y}) = 1)\Big)$. We distinguish the remaining clauses from (†):

· $(\nu_r(rep(x)) - \nu_r(rep(y)) \sim c \wedge (\nu_r(t_x) = 1 \vee \delta'(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta'(\tilde{y}) = 1))$. By construction of $\dot{s}_{r,\delta'}$, x and y respectively obtain from $\zeta(\dot{s}_{r,\delta'}, r, x, \delta')$ and $\zeta(\dot{s}_{r,\delta'}, r, y, \delta')$ the following values. Firstly x:

1. $\nu_r(rep(x))$, if there exists an edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_j)$ for some $Y \in \mathcal{EC}_{\mathcal{N}}$, such that the transformed version of $\mathcal{A}_j$ is located at ℓ in $\dot{s}_{r,\delta'}$ and $\nu_r(t_x) = 1$ or,
2. $\nu_s(x)$, if the transformed version of $\mathcal{A}_j$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting x.

Secondly y:

1. $\nu_r(rep(y))$, if there exists an edge $(\ell, \alpha, \varphi, \langle y := 0\rangle, \ell') \in SimpEdges_W(\mathcal{A}_k)$ for some $W \in \mathcal{EC}_{\mathcal{N}}$, such that the transformed version of $\mathcal{A}_k$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_y) = 0$ and $\delta'(\tilde{y}) = 1$ or, that automaton is located at ℓ in $\dot{s}_{r,\delta'}$ and $\nu_r(t_y) = 1$.

Thus, $\dot{s}_{r,\delta'} \models x - y \sim c$.

· $(0 - \nu_r(rep(y)) \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta'(\tilde{x}) = 1) \wedge (\nu_r(t_y) = 1 \vee \delta'(\tilde{y}) = 1))$. By construction of $\dot{s}_{r,\delta'}$, x and y respectively obtain from $\zeta(\dot{s}_{r,\delta'}, r, x, \delta')$ and $\zeta(\dot{s}_{r,\delta'}, r, y, \delta')$ the following values. Firstly x:

1. 0, if there exists a simple edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_j)$ for some $Y \in \mathcal{EC}_{\mathcal{N}}$, such that the transformed version of $\mathcal{A}_j$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_x) = 0$ and $\delta'(\tilde{x}) = 0$ or,
2. $\nu_s(x)$ if the transformed version of $\mathcal{A}_j$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting x.

Secondly y:

1. $\nu_r(rep(y))$, if there exists an edge $(\ell, \alpha, \varphi, \langle y := 0\rangle, \ell') \in SimpEdges_W(\mathcal{A}_k)$ for some $W \in \mathcal{EC}_\mathcal{N}$, such that the transformed version of $\mathcal{A}_k$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_y) = 0$ and $\delta'(\tilde{y}) = 1$ or, that automaton is located at ℓ in $\dot{s}_{r,\delta'}$ and $\nu_r(t_y) = 1$.

Thus, $\dot{s}_{r,\delta'} \models x - y \sim c$.

* $\delta'(\tilde{x}) = 1$ and $\delta'(\tilde{y}) = 0$. This case is symmetrical to the previous one.
* $\delta'(\tilde{x}) = 0$ and $\delta'(\tilde{y}) = 0$. For this case we discuss only the clause $(0 \sim c \wedge \neg(\nu_r(t_x) = 1) \wedge \neg(\delta'(\tilde{x}) = 1) \wedge \neg(\nu_r(t_y) = 1) \wedge \neg(\delta'(\tilde{y}) = 1))$, the discussion of the remaining clauses is similar as in the previous cases. By construction of $\dot{s}_{r,\delta'}$, x and y respectively obtain from $\zeta(\dot{s}_{r,\delta'}, r, x, \delta')$ and $\zeta(\dot{s}_{r,\delta'}, r, y, \delta')$ the following values. Firstly x:
  1. 0, if there exists a simple edge $(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_j)$ for some $Y \in \mathcal{EC}_\mathcal{N}$, such that the transformed version of $\mathcal{A}_j$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_x) = 0$ and $\delta'(\tilde{x}) = 0$ or,
  2. $\nu_s(x)$ if the transformed version of $\mathcal{A}_j$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting x.

  Secondly y:
  1. 0, if there exists a simple edge $(\ell, \alpha, \varphi, \langle y := 0\rangle, \ell') \in SimpEdges_W(\mathcal{A}_k)$ for some $W \in \mathcal{EC}_\mathcal{N}$, such that the transformed version of $\mathcal{A}_k$ is located at ℓ' in $\dot{s}_{r,\delta'}$, $\nu_r(t_y) = 0$ and $\delta'(\tilde{y}) = 0$ or,
  2. $\nu_s(y)$ if the transformed version of $\mathcal{A}_k$ in $\dot{s}_{r,\delta'}$ is neither located at origin nor at destination location of a simple edge resetting y.

– Induction step i. $CF = \neg CF_1$. We show that $r \models_{\delta'} \Omega_0(\neg CF_1) \implies \dot{s}_{r,\delta'} \models_{\delta'} \neg CF_1$.

$r \models_{\delta'} \Omega_0(\neg CF_1)$
$\iff$ By Definition 18 and Ω_0.
$\neg(r \models_{\delta'} \Omega_0(CF_1))$
$\implies$ By induction assumption.
$\neg(\dot{s}_{r,\delta'} \models_{\delta'} CF_1)$
$\iff$ By Definition 18.
$\dot{s}_{r,\delta'} \models_{\delta'} \neg CF_1$

– Induction step ii. $CF = CF_1 \wedge CF_2$. We show that $r \models_{\delta'} \Omega_0(CF_1 \wedge CF_2) \implies \dot{s}_{r,\delta'} \models_{\delta'} CF_1 \wedge CF_2$.

$r \models_{\delta'} \Omega_0(CF_1 \wedge CF_2)$
$\iff$ By Ω_0 Definition 28.
$r \models_{\delta'} \Omega_0(CF_1) \wedge \Omega_0(CF_2)$
$\iff$ By Definition 18.
$r \models_{\delta'} \Omega_0(CF_1) \wedge r \models_{\delta'} \Omega_0(CF_2)$
$\implies$ By induction assumption.
$\dot{s}_{r,\delta'} \models_{\delta'} CF_1 \wedge \dot{s}_{r,\delta'} \models_{\bar{\delta}} CF_2$
$\iff$ By Definition 18.
$\dot{s}_{r,\delta'} \models_{\delta'} CF_1 \wedge CF_2$

– Induction step iii. $CF = \exists\, \theta_1, \ldots, \theta_n \bullet CF$. We show that $r \models_{\delta'} \Omega_0(\exists\, \theta_1, \ldots, \theta_n \bullet CF) \implies \dot{s}_{r,\delta'} \models_{\delta'} \exists\, \theta_1, \ldots, \theta_n \bullet CF$.

$r \models_{\delta'} \Omega_0(\exists\, \theta_1, \ldots, \theta_n \bullet CF)$
$\iff$ By Ω_0 Definition 28.
$r \models_{\delta'} \exists\, \theta_1, \ldots, \theta_n \bullet \Omega_0(CF)$
$\iff$ By Definition 18.
$\exists\, \bar{d}_1, \ldots, \bar{d}_m \in \{0,1\} \bullet$
$r \models_{\delta'[\theta_i := \bar{d}_i | 1 \leq i \leq m]} \Omega_0(CF)$
$\implies$ By induction assumption.
$\exists\, \bar{d}_1, \ldots, \bar{d}_m \in \{0,1\} \bullet$
$s \models_{\delta[\theta_i := \bar{d}_i | 1 \leq i \leq m]} CF$
$\iff$ By Definition 18.
$s \models_{\delta} \exists\, \theta_1, \ldots, \theta_n \bullet CF$

□

The following proposition is used to construct and reach a configuration r' of a given transformed network $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$, such that from r' delay transitions greater or equal than 0 time units are possible. The intuition is as follows, we characterise a starting configuration r, such that from r we are able to sequentially "pull" resetters to their respective $\ell_{nst\mathcal{R}_Y}$-locations, and later synchronously all resetters are "pulled" to their respective initial locations.

Proposition 3

Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}} = \{Y_1, \ldots, Y_n\}$. Let $\mathcal{EC}_{\mathcal{N}}^{prio} = \langle Y_1, \ldots, Y_n \rangle$ be a list of all elements of $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. Let $\phi := \forall\, Y \in \mathcal{EC}_{\mathcal{N}} \bullet (\ell_{\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y} \wedge \sum_{x \in Y} t_x = 0 \wedge prio_Y = 1) \vee (\ell_{\mathcal{R}_Y} = \ell_{ini\mathcal{R}_Y} \wedge (\sum_{x \in Y} t_x = |Y| \vee \sum_{x \in Y} t_x = 0) \wedge prio_Y = 0)$. Let $r = (\langle \ell_1, \ldots, \ell_m, \ell_{\mathcal{R}_{Y_1}}, \ldots, \ell_{\mathcal{R}_{Y_n}} \rangle, \nu_r) \in Conf(\mathcal{N}')$ be a configuration such that $r \models \phi$. Then there is a sequence of transitions $\sigma : r \xrightarrow{\lambda}{}^* r'$, such that $r' \models \forall\, Y \in \mathcal{EC}_{\mathcal{N}}, x \in Y \bullet \ell_{\mathcal{R}_Y} = \ell_{ini\mathcal{R}_Y} \wedge t_x = 1 \wedge prio_Y = 0$.

Proof. To construct σ we proceed as follows:

- Let $S_1 = \{Y \in \mathcal{EC}_{\mathcal{N}} \mid r \models \forall\, x \in Y \bullet \ell_{\mathcal{R}_Y} = \ell_{ini\mathcal{R}_Y} \wedge t_x = 0 \wedge prio_Y = 0\}$. Note that S_1 may be empty.
- Set $\sigma := r$ and execute the following algorithm while $S_1 \neq \emptyset$:
  1. Pick $W \in S_1$, such that W has a higher index in $\mathcal{EC}_{\mathcal{N}}^{prio}$ than any other element in S_1.
  2. Consider the edge sending on the urgent broadcast channel u_W, $e_W = (\ell_{ini\mathcal{R}_W}, u_W!, \varphi_W, \langle prio_W := 0 \rangle, \ell_{nst\mathcal{R}_W}) \in E(\mathcal{R}_W)$, (by Algorithm 27 there are no edges listening on u_W) and propose the configuration $r' = (\vec{\ell}_r[\ell_{\mathcal{R}_W} := \ell_{nst\mathcal{R}_W}], \nu_r[\langle prio_W := 0 \rangle])$.
     Note that $\varphi_W = (\sum_{w \in W} t_w = 0)$. Recall that $W \in S_1$ if and only if $r \models \forall\, x \in W \bullet \ell_{\mathcal{R}_W} = \ell_{ini\mathcal{R}_W} \wedge t_x = 0 \wedge prio_W = 0$. Thus $\nu_r \models \varphi_W$. Note as well that $\nu_r[\langle prio_W := 0 \rangle] \models I(\ell_{nst\mathcal{R}_W})$ because $I(\ell_{nst\mathcal{R}_W})$ is the invariant of an urgent location constraining the lapse of time with the help of a fresh clock, and if invariants of other locations are satisfied in r then they are still satisfied in r' since only $\mathcal{R}_W$ changes location in r'. Hence, e_W is enabled.

Recall the edge $e = (\ell_{ini\mathcal{R}_W}, reset_W?, \varphi, \vec{r}, \ell_{nst\mathcal{R}_W}) \in E(\mathcal{R}_W)$ which is not enabled because there are no transformed simple edges sending on $reset_W$ enabled at the current time, which is deduced from $r \models \forall\, x \in W \bullet \ell_{\mathcal{R}_W} = \ell_{ini\mathcal{R}_W} \wedge t_x = 0 \wedge prio_W = 0$. Since W has a higher index in $\mathcal{EC}_{\mathcal{N}}^{prio}$ than any other element in S_1, thus, in $\mathcal{N}'$ the transition justified by e_W has the highest priority at the current time (see the prioritisation mechanism implemented by Algorithm $\mathcal{K}$, Definition 27). Hence, $r \xrightarrow{u_W} r'$.

3. Set $S_1 := S_1 \setminus \{W\}$, $r := r'$ and $\sigma := \sigma \xrightarrow{u_W} r'$.

Note that the above algorithm terminates because we leave its loop by decrementing the size of S_1 in each iteration until $S_1 = \emptyset$.

- Let $S_2 = \{Y \in \mathcal{EC}_{\mathcal{N}} \mid r \models \forall\, x \in Y \bullet \ell_{\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y} \wedge t_x = 0 \wedge prio_Y = 1\}$. Note that S_1 and S_2 are empty if $r \models \forall Y \in \mathcal{EC}_{\mathcal{N}}, x \in Y \bullet \ell_{\mathcal{R}_Y} = \ell_{ini\mathcal{R}_Y} \wedge t_x = 1 \wedge prio_Y = 0$.

  If S_1 and S_2 are empty then we use r to construct r', i.e. $r' := r$, where $r' \models \forall Y \in \mathcal{EC}_{\mathcal{N}}, x \in Y \bullet \ell_{\mathcal{R}_Y} = \ell_{ini\mathcal{R}_Y} \wedge t_x = 1 \wedge prio_Y = 0$, and we propose $\sigma \xrightarrow{0} r'$. Otherwise, we assume that S_2 is not empty, and we consider the equivalence classes in the set $S_2 = \{W_1, \ldots, W_{\dot{n}}\}$ and the two sets of edges: $\hat{P} = \{\hat{e}_{W_1} = (\ell_{nst\mathcal{R}_{W_1}}, return!, \hat{\varphi}_{W_1}, \hat{\vec{r}}_{W_1}, \ell_{ini\mathcal{R}_{W_1}}) \in E(\mathcal{R}_{W_1}), \ldots, \hat{e}_{W_{\dot{n}}} = (\ell_{nst\mathcal{R}_{W_{\dot{n}}}}, return!, \hat{\varphi}_{W_{\dot{n}}}, \hat{\vec{r}}_{W_{\dot{n}}}, \ell_{ini\mathcal{R}_{W_{\dot{n}}}}) \in E(\mathcal{R}_{W_{\dot{n}}})\}$, and $\dot{P} = \{\dot{e}_{W_1} = (\ell_{nst\mathcal{R}_{W_1}}, return?, \dot{\varphi}_{W_1}, \dot{\vec{r}}_{W_1}, \ell_{ini\mathcal{R}_{W_1}}) \in E(\mathcal{R}_{W_1}), \ldots, \dot{e}_{W_{\dot{n}}} = (\ell_{nst\mathcal{R}_{W_{\dot{n}}}}, return?, \dot{\varphi}_{W_{\dot{n}}}, \dot{\vec{r}}_{W_{\dot{n}}}, \ell_{ini\mathcal{R}_{W_{\dot{n}}}}) \in E(\mathcal{R}_{W_{\dot{n}}})\}$ of $\mathcal{N}'$. Execute the following steps:

  1. Pick $W \in S_2$, such that W has a higher index in $\mathcal{EC}_{\mathcal{N}}^{prio}$ than any other element in S_2.
  2. Let W be the equivalence class of the resetter sending on the channel $return$, and let $\{W_1, \ldots, W_{\dot{n}-1}\} := \{W_1, \ldots, W_{\dot{n}}\} \setminus \{W\}$ be the set of equivalence classes of resetters listening on $return$. Propose $r' = (\vec{l}_r[\ell_{\mathcal{R}_W} := \ell_{ini\mathcal{R}_W}][\ell_{\mathcal{R}_{W_1}} := \ell_{ini\mathcal{R}_{W_1}}] \ldots [\ell_{\mathcal{R}_{W_{\dot{n}-1}}} := \ell_{ini\mathcal{R}_{W_{\dot{n}-1}}}], \nu_r[\hat{\vec{r}}_W][\dot{\vec{r}}_{W_1}] \ldots [\dot{\vec{r}}_{W_{\dot{n}-1}}])$.

  Note that $\hat{\varphi}_W = (\sum_{w \in W} t_w = 0 \wedge blk(\mathcal{EC}_{\mathcal{N}}) \wedge prties(\mathcal{EC}_{\mathcal{N}}))$ and that $\nu_r \models \hat{\varphi}_W$ by the following reasons:

  1. $prio_W = 1$, W has a higher index in $\mathcal{EC}_{\mathcal{N}}^{prio}$ than any other element in S_2, and $prio_{\bar{W}} = 0$, for each $\bar{W} \notin S_2$ with a higher index than W in $\mathcal{EC}_{\mathcal{N}}^{prio}$, since the resetter $\mathcal{R}_{\bar{W}}$ is located at $\ell_{ini\mathcal{R}_{\bar{W}}}$ in r (by ϕ, $\mathcal{R}_{\bar{W}}$ cannot be located at $Tlock_{\bar{W}}$ in r), therefore, $\nu_r \models prties(\mathcal{EC}_{\mathcal{N}})$,
  2. $W \in S_2$ if and only if $r \models \forall\, x \in W \bullet \ell_{\mathcal{R}_W} = \ell_{nst\mathcal{R}_W} \wedge t_x = 0 \wedge prio_W = 1$, therefore, $\nu_r \models \sum_{w \in W} t_w = 0$,
  3. $\ell_{r,\mathcal{R}_{\bar{W}}} = \ell_{nst\mathcal{R}_{\bar{W}}}$ and $\nu_r \models \sum_{w \in \bar{W}} t_w = 0$, for each $\bar{W} \in S_2$, while $\ell_{r,\mathcal{R}_{\dot{W}}} = \ell_{ini\mathcal{R}_{\dot{W}}}$ and $\nu_r \models \sum_{w \in \dot{W}} t_w = |\dot{W}|$, for each $\dot{W} \notin S_2$, therefore, $\nu_r \models blk(\mathcal{EC}_{\mathcal{N}})$.

  Similarly, $\dot{\varphi}_{\bar{W}} = (\sum_{w \in \bar{W}} t_w = 0 \wedge blk(\mathcal{EC}_{\mathcal{N}}))$, for each equivalence class $\bar{W} \in \{W_1, \ldots, W_{\dot{n}-1}\}$ and $\nu_r \models \dot{\varphi}_{\bar{W}}$ by reasons 2 and 3. Note as

well that $\nu_r[\hat{\vec{r}}_W][\dot{\vec{r}}_{W_1}]\dots[\dot{\vec{r}}_{W_{\dot{n}-1}}] \models \bigwedge_{Y\in\{W,W_1,\dots,W_{\dot{n}-1}\}} I(\ell_{ini_Y})$ because $\ell_{ini_{\mathcal{R}_W}}, \ell_{ini_{\mathcal{R}_{W_1}}}, \dots, \ell_{ini_{\mathcal{R}_{W_{\dot{n}-1}}}}$ have trivial invariants, i.e. $true$, and if invariants of other locations are satisfied in r then they are still satisfied in r' since only $\mathcal{R}_W, \mathcal{R}_{W_1}, \dots, \mathcal{R}_{W_{\dot{n}-1}}$ changed locations in r'. Hence, the edges $\hat{e}_W, \dot{e}_{W_1}, \dots, \dot{e}_{W_{\dot{n}-1}}$ are enabled.

Recall each edge $e = (\ell_{nst_{\mathcal{R}_W}}, \tau, go(\mathcal{EC}_\mathcal{N}), \langle\rangle, Tlock_W) \in E(\mathcal{R}_W)$, with $W \in S_2$, which is not enabled because $\nu_r \not\models go(\mathcal{EC}_\mathcal{N})$, in particular, because in r no token related to a clock in W has value 1 (which is required by $go(\mathcal{EC}_\mathcal{N})$) as defined for each equivalence class in S_2. Thus, by the semantics of timed automata (Definition 16), the broadcast transition justified by taking edges $\hat{e}_W, \dot{e}_{W_1}, \dots, \dot{e}_{W_{\dot{n}-1}}$ occurs, i.e. $\sigma := \sigma \xrightarrow{\tau} r'$. Note that by construction of r', $r' \models \forall Y \in \mathcal{EC}_\mathcal{N}, x \in Y \bullet \ell_{\mathcal{R}_Y} = \ell_{ini_{\mathcal{R}_Y}} \wedge t_x = 1 \wedge prio_Y = 0$. □

The following proposition is used to construct and reach a configuration r' of a given transformed network $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_\mathcal{N})$, which results from taking all simple edges enabled at the current time.

In this proposition we consider cases where time is stopped (and no delay transitions greater than 0 are possible) either at origin or at destination locations of simple edges. The intuition for this proposition is as follows, we characterise a starting configuration r, such that from r we sequentially reach configurations resulting from taking all simple edges enabled at the current time. In this case we reach the configuration r' where all resetters related to those enabled edges, are located at their respective $\ell_{nst_{\mathcal{R}_Y}}$-locations. Moreover, if we recognise that time has been stopped in $\mathcal{N}$ then we introduce the respective configurations and transitions, such that we reach the configuration r' where time has been stopped as well in $\mathcal{N}'$.

Proposition 4

Let $\mathcal{N} = \{\mathcal{A}_1, \dots, \mathcal{A}_q\}$ be a network with $\mathcal{EC}_\mathcal{N} = \{Y_1, \dots, Y_n\}$. Let $\mathcal{EC}^{prio}_\mathcal{N} = \langle Y_1, \dots, Y_n\rangle$ be a list of all elements of $\mathcal{EC}_\mathcal{N}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}^{prio}_\mathcal{N})$. Let $CF \in CF_\mathcal{N}$ be a property over $\mathcal{N}$. Let $s, s' \in Conf(\mathcal{N})$ and $r \in Conf(\mathcal{N}')$ be configurations, such that $r \in QE(s)$, $s \in StableConf_Y$, and the transition $s \xrightarrow{\lambda}_{E_Y} s'$ is justified by the set E_Y of simple edges wrt. some $Y \in \mathcal{EC}_\mathcal{N}$, and $s' \models_\delta CF$.

Then there exists a configuration r', and a sequence of transitions $\sigma : r \xrightarrow{\lambda}{}^* r'$, such that $r' \in QE(s')$.

Proof. Let $S = \{Y \in \mathcal{EC}_\mathcal{N} \mid s \xrightarrow{\lambda}_{E_Y} s' \wedge s \in StableConf_Y\}$. We know that S is not empty, and at least contains $Y \in \mathcal{EC}_\mathcal{N}$ since the transition from s to s' is justified by the set E_Y of simple edges wrt. Y.

- Set $\sigma := r$ and execute the following steps while $S \neq \emptyset$:

  1. Pick $W \in S$, such that W has a higher index in $\mathcal{EC}^{prio}_\mathcal{N}$ than the index of any other equivalence class in S.
  2. Let $\bar{P} := \{(\ell, \alpha, \varphi, \vec{r}, \ell') \in SimpEdges_W(\mathcal{N}) \mid \exists i \in \mathbb{N} \bullet \ell_{s,i} = \ell\}$, i.e. the set of simple edges wrt. W whose origin locations are the current locations in s of some automata of $\mathcal{N}$.
     Let $P := \{(\ell, \alpha, \varphi, \vec{r}, \ell') \in SimpEdges_W(\mathcal{N}) \mid \exists i \in \mathbb{N} \bullet \ell_{s,i} = \ell \wedge \nu_s \models \varphi \wedge \nu_s[\vec{r}\,] \models I(\ell')\}$, i.e. the set of simple edges wrt. W

enabled in s. By Definition 22 there is a delay $d > 0$ at the origin location ℓ of each simple edge e before e can be taken, and by the same definition each ℓ has a unique outgoing edge, namely, edge e. Therefore, we are able to obtain in P all simple edges wrt. W that are enabled in configuration s at the current time.

Note that $P \subseteq \bar{P}$, since there may be simple edges in $\bar{P}$ whose guards are not satisfied by ν_s.

Recall that the set $P = \{e_1 = (\ell_1, \alpha_1, \varphi_1, \vec{r}_1, \ell'_1), \ldots,$
$e_m = (\ell_m, \alpha_m, \varphi_m, \vec{r}_m, \ell'_m)\}$ has corresponding sets $\hat{P} = \{\hat{e}_1 = (\hat{\ell}_1, reset_W!, \hat{\varphi}_1, \hat{\vec{r}}_1, \hat{\ell}'_1), \ldots, \hat{e}_m = (\hat{\ell}_m, reset_W!, \hat{\varphi}_m, \hat{\vec{r}}_m, \hat{\ell}'_m)\}$ and $\dot{P} = \{\dot{e}_1 = (\dot{\ell}_1, reset_W?, \dot{\varphi}_1, \dot{\vec{r}}_1, \dot{\ell}'_1), \ldots,$
$\dot{e}_m = (\dot{\ell}_m, reset_W?, \dot{\varphi}_m, \dot{\vec{r}}_m, \dot{\ell}'_m)\}$ (where $\hat{\ell}_i = \dot{\ell}_i$ and $\hat{\ell}'_i = \dot{\ell}'_i$, for each $1 \leq i \leq m$), which together with
$e_{\mathcal{R}_W} = (\ell_{ini\mathcal{R}_W}, reset_Y?, true, \langle prio_Y := 1\rangle, \ell_{nst\mathcal{R}_W}) \in L(\mathcal{R}_W)$ are obtained by applying Algorithm $\mathcal{K}$ on $\mathcal{N}$ (see transformation of simple edges and construction of resetters in Definition 27).

Since $s \in StableConf_W$ and $r \in QE(s)$ we distinguish two situations in which the resetter $\mathcal{R}_W$ can be located in r:

(a) $\mathcal{R}_W$ is not located at $\ell_{ini\mathcal{R}_W}$. Set $r' := r$. By Proposition 2, $r' \models_\delta \Omega(CF)$. We propose a 0-delay transition. Thus, by the semantics of extended timed automata (Definition 16), $r \xrightarrow{0} r'$. Set $r := r'$ and $\sigma := \sigma \xrightarrow{0} r'$. We take r' as constructed and we show that $r' \in QE(s')$.
   - Values of integer variables existing in $\mathcal{N}$ are neither updated from s to s' (simple edges reset only clocks) nor from r to r' since $r' = r$. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, s' and r' satisfy Rule (6.2.1).
   - s' and r' (both) satisfy either Rule (6.0.2a) or Rule (6.0.2b). If s' and r' satisfy Rule (6.0.2a) then automaton $\mathcal{A}'_j$, for some $j \in \{1, \ldots, q\}$, and its original version are located at the same location, i.e. $\ell_{r',j} = \ell_{s',j}$, and values of $rep(x)$ and t_x, with $x \in \mathcal{X}(\mathcal{A}_i)$, encode in r' the value of x in s, thus, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$.
     Otherwise, s' and r' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$; the automaton $\mathcal{A}'_j$ is located in r' at the destination location of a transformed simple edge e wrt. Y, while the origin location of e is the current location of $\mathcal{A}_j$ in s'; $\nu_{r'}(rep(x)) = \nu_s(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
   - The book-keeping variable $s_Y^{\mathcal{A}_i}$, with $1 \leq i \leq q$ and $Y \in \mathcal{EC}_\mathcal{N}$, is not updated from r to r', hence, has in r' value 1 if $\mathcal{A}'_i$ is located at the origin location of a transformed simple edge wrt. Y, otherwise value 0. Thus, r' satisfies Rule (6.2.3).
   - The variable $prio_Y$, $Y \in \mathcal{EC}_\mathcal{N}$, is not updated from r to r', hence, $prio_Y$ has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

(b) $\mathcal{R}_W$ is located at $\ell_{ini\mathcal{R}_W}$. Proceed as follows.

i. Let $k \in \{1,\dots,m\}$ be the index of the automaton in $\hat{P}$ sending on the channel $reset_W$, and $\{1,\dots,m-1\} = \{1,\dots,m\}\setminus\{k\}$ be the set of indices of automata in $\dot{P}$ receiving on $reset_W$. Propose configuration $r' = (\vec{\ell_r}[\ell_{r,k} := \hat{\ell}'_k][\ell_{r,1} := \dot{\ell}'_1]\dots[\ell_{r,m-1} := \dot{\ell}'_{m-1}][\ell_{\mathcal{R}_W} := \ell_{nst\mathcal{R}_W}],$ $\nu_r[\vec{\hat{r}}_k][\vec{\dot{r}}_1]\dots[\vec{\dot{r}}_{m-1}][prio_W \mapsto 1])$. By Proposition 2, $r' \models_\delta \Omega(CF)$.
Note that by construction of $\mathcal{N}'$, $\hat{\varphi}_i = \dot{\varphi}_i$, for each $1 \le i \le m$, and that each $\hat{\varphi}_i$ is equivalent to φ_i. Since $s \in StableConf_Y$ and $r \in QE(s)$, by Rule (6.0.2a), the value of each clock $x \in W$ in s is equal to the value that the representative $rep(x)$ and token t_x encode in r for x. Hence, $\nu_r \models \hat{\varphi}_k \wedge \dot{\varphi}_1 \wedge \dots \wedge \dot{\varphi}_{m-1}$, and $\nu_r[\vec{\hat{r}}_k][\vec{\dot{r}}_1]\dots[\vec{\dot{r}}_{m-1}]\,[prio_W \mapsto 1] \models I(\hat{\ell}'_k) \wedge I(\dot{\ell}'_1) \wedge \dots \wedge I(\dot{\ell}'_{m-1}) \wedge I(\ell_{nstW})$, because $I(\hat{\ell}'_k) = I(\Gamma(\ell'_k))$, $I(\dot{\ell}'_1) = I(\Gamma(\ell'_1)),\dots, I(\dot{\ell}'_{m-1}) = I(\Gamma(\ell'_{m-1}))$ and $I(\ell_{nstW})$ is the invariant of an urgent location constraining the lapse of time with the help of a fresh clock. Hence, the edges $\hat{e}_k, \dot{e}_1, \dots, \dot{e}_{m-1}, e_{\mathcal{R}_W}$ are enabled. Thus, by the semantics of extended timed automata (Definition 16), $r \xrightarrow{\tau} r'$.
We take r' as constructed and we show that $r' \in QE(s')$.
   - Values of integer variables existing in $\mathcal{N}$ are not updated from r to r' (neither simple edges nor transformed simple ones update integer variables existing in $\mathcal{N}$). Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' satisfies Rule (6.2.1).
   - Both configurations s' and r' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If s' and r' satisfy Rule (6.0.2a) then for the automaton $\mathcal{A}'_j$ (and its original version), for some $j \in \{1,\dots,q\}$, i.e. an automaton which did not change location from r to r' (holds as well for the automaton which justifies the transition from s to s'), $\ell_{r',j} = \ell_{s',j}$ and $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$, with $x \in \mathcal{X}(\mathcal{A}_j)$. Otherwise, s' and r' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$; the automaton $\mathcal{A}'_j$ is located in r' at the destination location of a transformed simple edge e wrt. Y, while the origin location of e is the current location of $\mathcal{A}_j$ in s'; $\nu_{r'}(rep(x)) = \nu_s(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
   - Values of book-keeping variables $s_W^{\mathcal{A}_i}$, for each index $i \in \{k, 1,\dots,m-1\}$, are set to 1 in r' if the destination locations of the respective edges that update $s_W^{\mathcal{A}_i}$ are origin locations of transformed simple edges wrt. W, otherwise are set to 0.
   Let $P' := \bar{P} \setminus P$, i.e. the set of simple edges wrt. W which are not enabled since their guards are not satisfied by ν_s, and whose origin locations are the current locations in s of some automata of $\mathcal{N}$. By construction

of $\mathcal{N}'$ there exists a set P'' corresponding to P'. Book-keeping variables reset by edges in P'' have value 1 since they are in r' still located at the origin locations of edges in P''.
Moreover, book-keeping variables related to $Y \in \mathcal{EC}_{\mathcal{N}} \setminus \{W\}$ have in r' value 1 if the respective transformed automata updating these variables are located at locations which are origin locations of transformed simple edges wrt. Y, otherwise have value 0. Thus, r' satisfies Rule (6.2.3).

– Variable $prio_W$ is set to 1 since the resetter $\mathcal{R}_W$ is located at $\ell_{nst\mathcal{R}_W}$ in r'. Variable $prio_Y$, $Y \in \mathcal{EC}_{\mathcal{N}} \setminus \{W\}$, has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

3. Set $S := S \setminus \{W\}$. Set $r := r'$ and $\sigma := \sigma \xrightarrow{\tau} r'$.

Note that the above algorithm terminates because we leave its loop by decrementing the size of S in each iteration until $S = \emptyset$.

Let $S = \{Y \in \mathcal{EC}_{\mathcal{N}} \mid s \xrightarrow{\lambda}_{E_Y} s' \wedge s \in StableConf_Y\}$. We extend σ by executing the following steps while $S \neq \emptyset$:

1. Pick $W \in S$, such that W has a higher index in $\mathcal{EC}_{\mathcal{N}}^{prio}$ than the index of any other equivalence class in S.

2. Let $\bar{P} := \{(\ell, \alpha, \varphi, \vec{r}, \ell') \in SimpEdges_W(\mathcal{N}) \mid \exists\, i \in \mathbb{N} \bullet \ell_{s,i} = \ell\}$, i.e. the set of simple edges wrt. W whose origin locations are the current locations in s of some automata of $\mathcal{N}$.

3. Let $P := \{(\ell, \alpha, \varphi, \vec{r}, \ell') \in SimpEdges_W(\mathcal{N}) \mid \exists\, i \in \mathbb{N} \bullet \ell_{s,i} = \ell \wedge \nu_s \models \varphi \wedge \nu_s[\vec{r}\,] \models I(\ell')\}$, i.e. the set of simple edges wrt. W enabled in s.

4. Let $P' := \bar{P} \setminus P$, i.e. the set of simple edges wrt. W which are not enabled since their guards are not satisfied by ν_s, and whose origin locations are the current locations in s of some automata of $\mathcal{N}$.
If $P' \neq \emptyset$ then:

   - By construction of $\mathcal{N}'$ there exists the set of edges P'' corresponding to P'. Consider the following edge of the resetter $\mathcal{R}_W$, $e_W = (\ell_{nst\mathcal{R}_W}, \tau, go(\mathcal{EC}_{\mathcal{N}}), \langle prio_W := 0\rangle, Tlock_W)$. Recall that the current location of $\mathcal{R}_W$ in r is $\ell_{nst\mathcal{R}_W}$. Propose $r' := (\vec{\ell}_r[\ell_{\mathcal{R}_W} := Tlock_W], \nu_r[prio_W := 0])$. Note that $r' \in QE(s')$. By Proposition 2, $r' \models_\delta \Omega(CF)$.
     Note that the guard $go(\mathcal{EC}_{\mathcal{N}}) = prties(\mathcal{EC}_{\mathcal{N}}) \wedge \bigvee_{x \in W} sum(x)$ of e_W is satisfied by r by the following reasons:
     (a) $prio_W = 1$ (the current location of $\mathcal{R}_W$ in r is $\ell_{nst\mathcal{R}_W}$), W has a higher index in $\mathcal{EC}_{\mathcal{N}}^{prio}$ than any other element in S, and $prio_{\bar{W}} = 0$, for each $\bar{W} \notin S$ with a higher index than W in $\mathcal{EC}_{\mathcal{N}}^{prio}$, since the resetter $\mathcal{R}_{\bar{W}}$ in r is either located at $\ell_{ini\mathcal{R}_{\bar{W}}}$ (because no simple edge wrt. $\bar{W}$ is enabled at the current time), or at $Tlock_{\bar{W}}$ (because a transition to that location was possible and has been taken at the current time). Therefore, $\nu_r \models prties(\mathcal{EC}_{\mathcal{N}})$.

(b) Since $P'' \neq \emptyset$, then there is $e \in P''$ and the value of the variables occurring in the reset vector of e is as follows: (a) $s_W^{\mathcal{A}_i} = 1$, for some $1 \leq i \leq m$, since $\mathcal{A}'_i$ is located at the origin location of e, (b) $t_x = 1$, for some $x \in \mathcal{X}(\mathcal{A}_i)$, since $s \in StableConf_W$, and wrt. automata $\mathcal{A}_i$ in s and $\mathcal{A}'_i$ in r', both configurations satisfy Rule (6.0.2a). Thus, $\nu_r \models \bigvee_{x \in W} sum(x)$.

Note that $I(Tlock_W)$ is the invariant of an urgent location constraining the lapse of time with the help of a fresh clock. Thus, $\nu_r[prio_W := 0] \models I(Tlock_W)$. Hence, e is enabled. Thus, by the semantics of extended timed automata (Definition 16), $r \xrightarrow{\tau} r'$. Set $r := r'$ and $\sigma := \sigma \xrightarrow{\tau} r'$.

5. Set $S := S \setminus \{W\}$. Set $r := r'$ and $\sigma := \sigma \xrightarrow{\tau} r'$.

Note that the above algorithm terminates because we leave its loop by decrementing the size of S in each iteration until $S = \emptyset$. □

In the following lemma we show that there exists a weak bisimulation between a network $\mathcal{N}$ with equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$, and its corresponding transformed network $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$.

Lemma 2 *Weak Bisimulation.*
Let $\mathcal{N}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements of $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{CF}_{\mathcal{N}}$ be the set of configuration formulas over $\mathcal{N}$. Then $\mathcal{N}$ is *weakly bisimilar* to $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$, i.e. there is a *weak bisimulation relation* $QE \subseteq Conf(\mathcal{N}) \times Conf(\mathcal{N}')$ such that:

1. $\forall\, s \in \mathcal{C}_{ini}(\mathcal{N})\ \exists\, r \in \mathcal{C}_{ini}(\mathcal{N}') \bullet r \in QE(s)$ and $\forall\, r \in \mathcal{C}_{ini}(\mathcal{N}')\ \exists\, s \in \mathcal{C}_{ini}(\mathcal{N}) \bullet r \in QE(s)$.
2. $\forall\, CF \in \mathcal{CF}_{\mathcal{N}} \forall\, r \in QE(s) \bullet s \models_{\delta} CF \implies r \models_{\delta} \Omega(CF)$.
3. $\forall\, CF \in \mathcal{CF}_{\mathcal{N}} \forall\, r \in QE(s) \bullet r \models_{\delta} \Omega(CF) \implies \exists\, \dot{s} \in Conf(\mathcal{N}) \bullet r \in QE(\dot{s}) \wedge \dot{s} \models_{\delta} CF$.
4. $\forall\, r \in QE(s) \bullet s \xrightarrow{\lambda} s' \implies \exists\, r \xrightarrow{\lambda}{}^{*}\ r' \bullet r' \in QE(s')$. We distinguish the transition $s \xrightarrow{\lambda} s'$ as follows:
   (a) transition is justified by a delay $\lambda = d \geq 0$,
   (b) $s \in \bigcap_{Y \in \mathcal{EC}_{\mathcal{N}}} StableConf_Y, s' \notin \bigcap_{Y \in \mathcal{EC}_{\mathcal{N}}} StableConf_Y$, and the transition is justified by a simple edge wrt. $Y \in \mathcal{EC}_{\mathcal{N}}$,
   (c) $s \notin \bigcap_{Y \in \mathcal{EC}_{\mathcal{N}}} StableConf_Y$ and the transition is justified by a simple edge wrt. $Y \in \mathcal{EC}_{\mathcal{N}}$,
   (d) transition is justified by a non-empty set of edges E, such that none of them is simple wrt. any $Y \in \mathcal{EC}_{\mathcal{N}}$.
5. $\forall\, CF \in \mathcal{CF}_{\mathcal{N}} \forall\, r \in QE(s) \bullet r \xrightarrow{\lambda} r' \wedge r' \models_{\delta'} \Omega_0(CF) \implies \exists\, s \xrightarrow{\lambda}{}^{*}\ s' \bullet r' \in QE(s')$.

   We distinguish the transition $r \xrightarrow{\lambda} r'$ as follows:
   (a) transition is justified by a delay $\lambda = d \geq 0$,
   (b) transition is justified by a non-empty set of edges E.

Proof.

Let $\mathcal{N} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a network with a set of equivalence classes of quasi-equal clocks $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{EC}_{\mathcal{N}}^{prio}$ be a list of all elements of $\mathcal{EC}_{\mathcal{N}}$. Let $\mathcal{CF}_{\mathcal{N}}$ be the set of configuration formulas over $\mathcal{N}$. Let $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{EC}_{\mathcal{N}}^{prio})$. We show that the binary relation QE is a *weak bisimulation relation* such that:

1. $\forall\, s \in \mathcal{C}_{ini}(\mathcal{N})\ \exists\, r \bullet r \in QE(s)$ and $\forall\, r \in \mathcal{C}_{ini}(\mathcal{N}')\ \exists\, s \bullet r \in QE(s)$. For the sets $\mathcal{C}_{ini}(\mathcal{N})$ and $\mathcal{C}_{ini}(\mathcal{N}')$ we have:
   $\mathcal{C}_{ini}(\mathcal{N}) = \emptyset \iff \mathcal{C}_{ini}(\mathcal{N}') = \emptyset$.

   The only candidates for valuations in the initial configuration are ν_{s_0} and ν_{r_0} which assign all clocks to 0 and all variables to their initial values. Then we have

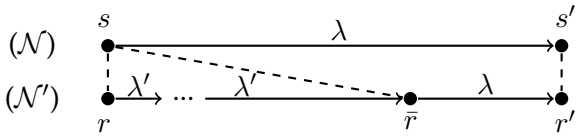
$\mathcal{C}_{ini}(\mathcal{N}) = \emptyset$
$\iff$ By Definition 16.
$\nu_{s_0} \not\models_\delta \bigwedge_{i=1}^{n} I(\ell_{ini,s_0,i})$
$\iff$ By Algorithm $\mathcal{K}$ (Definition 27).
$\nu_{r_0} \not\models_\delta \bigwedge_{i=1}^{n} \Gamma(I(\ell_{ini,s_0,i})) \wedge \bigwedge_{Y \in \mathcal{EC}_\mathcal{N}} I(\ell_{ini\mathcal{R}_Y})$
$\iff$ By Definition 16.
$\mathcal{C}_{ini}(\mathcal{N}') = \emptyset$

2. $\forall\, CF \in \mathcal{CF}_\mathcal{N} \forall r \in QE(s) \bullet s \models_\delta CF \implies r \models_\delta \Omega(CF)$. This claim holds by Proposition 2.

3. $\forall\, CF \in \mathcal{CF}_\mathcal{N} \forall r \in QE(s) \bullet r \models_\delta \Omega(CF) \implies \exists\, \dot{s} \in Conf(\mathcal{N}) \bullet r \in QE(\dot{s}) \wedge \dot{s} \models_\delta CF$. This claim holds by Proposition 2.

4. $\forall r \in QE(s) \bullet s \xrightarrow{\lambda} s' \implies \exists r \xrightarrow{\lambda}{}^{*} r' \bullet r' \in QE(s')$. We distinguish the transition $s \xrightarrow{\lambda} s'$ as follows:

   (a) transition is justified by a delay $\lambda = d \geq 0$.



   We show there exist $\bar{r}, r' \in Conf(\mathcal{N}')$, such that $r \xrightarrow{\lambda'}{}^{*} \bar{r} \xrightarrow{\lambda} r'$, $\bar{r} \in QE(s)$ and $r' \in QE(s')$. We distinguish two cases based on d, namely, $d = 0$ and $d > 0$.

   The first case $d = 0$ is trivial, since this transition in $\mathcal{N}$ is of the type $\langle \vec{\ell}_s, \nu_s \rangle \xrightarrow{0} \langle \vec{\ell}_s, \nu_s + 0 \rangle$, i.e. $s = s'$, and we propose $\bar{r}$, r' such that $r = \bar{r} = r'$. Thus, by the semantics of extended timed automata (Definition 16), $r \xrightarrow{0} \bar{r} \xrightarrow{0} r'$. Since $r \in QE(s)$, $s = s'$ and $r = \bar{r} = r'$, hence, $\bar{r} \in QE(s)$ and $r' \in QE(s')$.

   We show the second case $d > 0$.

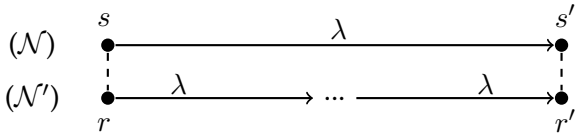
   - This transition is of the type $\langle \vec{\ell}_s, \nu_s \rangle \xrightarrow{d} \langle \vec{\ell}_s, \nu_s + d \rangle$. Hence, $s \in \bigcap_{Y \in \mathcal{EC}_\mathcal{N}} StableConf_Y$, otherwise, if $s \notin StableConf_Y$, for some $Y \in \mathcal{EC}_\mathcal{N}$, then quasi-equality does not hold for Y (see Lemma 1) after the delay d.
   - Since $s \in \bigcap_{Y \in \mathcal{EC}_\mathcal{N}} StableConf_Y$ and $r \in QE(s)$, we know that by Function QE (Definition 29), $r \models \phi$, with $\phi := \forall\, Y \in \mathcal{EC}_\mathcal{N} \bullet (\ell_{\mathcal{R}_Y} = \ell_{nst\mathcal{R}_Y} \wedge \sum_{x \in Y} t_x = 0 \wedge prio_Y = 1) \vee (\ell_{\mathcal{R}_Y} = \ell_{ini\mathcal{R}_Y} \wedge (\sum_{x \in Y} t_x = |Y| \vee \sum_{x \in Y} t_x = 0) \wedge prio_Y = 0)$, otherwise r would violate, in particular, rules (6.0.2a) and (6.0.2b).
   - Use Proposition 3 to construct and reach $\bar{r}$. We take $\bar{r}$ as constructed and we show that $\bar{r} \in QE(s)$.
     - Values of integer variables existing in $\mathcal{N}$ are not changed, since Proposition 3 uses only edges in resetters and these do not update integer variables of $\mathcal{N}$. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s and in $\bar{r}$, i.e. $\nu_s(v) = \nu_{\bar{r}}(v)$. Thus, $\bar{r}$ and s satisfy Rule (6.2.1).
     - Values of $rep(x)$ and t_x, for some $Y \in \mathcal{EC}_\mathcal{N}$ and for each $x \in Y$, have been updated to 0 and 1, respectively. Recall

that in $\bar{r}$ we have changed locations of resetters to their respective initial locations (other automata are located as in r). Thus, for each $1 \leq i \leq n$, $\ell_{s,i} = \ell_{\bar{r},i}$, values of $rep(x)$ and t_x, with $x \in \mathcal{X}(\mathcal{A}_i)$, encode in $\bar{r}$ the value of x in s. Thus, $\nu_s(x) = \nu_{\bar{r}}(rep(x)) \cdot \nu_{\bar{r}}(t_x)$. Hence, $\bar{r}$ and s satisfy Rule (6.0.2a).

- Values of book-keeping variables are not updated in configuration $\bar{r}$. Thus, the value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$ and for some $Y \in \mathcal{EC}_{\mathcal{N}}$, is 1 in $\bar{r}$ if and only if $\mathcal{A}'_i$ is located at the origin location of a transformed simple edge wrt. Y. Thus, $\bar{r}$ satisfies Rule (6.2.3).
- All resetters are now located at their initial locations and their respective variables $prio_Y$ have value 0. Thus $\bar{r}$ satisfies (6.2.4).

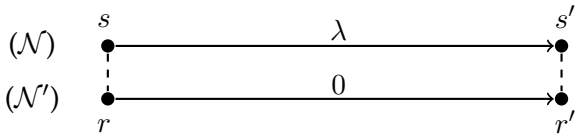
- Propose $r' := \langle \vec{\ell}_{\bar{r}}, \nu_{\bar{r}} + d \rangle$, where all clocks in $\mathcal{X}(\mathcal{N}')$ have advanced d time units. The configuration r' still satisfies the rules of QE. Thus, $r' \in QE(s')$.
- Note that the invariants of the current locations of transformed automata in $\bar{r}$ are invariants equivalent to the respective original ones of current locations of automata in s', i.e. $I(\ell_{\bar{r},i}) = \Gamma(I(\ell_{s,i}))$, for each $1 \leq i \leq n$, which have been introduced by Algorithm 27. That algorithm introduces as well resetter automata, and since those automata are in $\bar{r}$ currently located at their respective initial locations, their invariants are trivially satisfied. In other words, $\nu_{\bar{r}} + d \models \bigwedge_{i=1}^{n} \Gamma(I(\ell_{s',i})) \wedge \bigwedge_{Y \in \mathcal{EC}_{\mathcal{N}}} I(\ell_{\mathcal{R}_Y})$. Thus, by the semantics of extended timed automata (Definition 16), $\bar{r} \xrightarrow{d} r'$.

(b) $s \in StableConf_Y, s' \in Conf(\mathcal{N}')$, for some $Y \in \mathcal{EC}_{\mathcal{N}}$, and the transition is justified by a set of simple edges E_Y.



We show there exists $r' \in Conf(\mathcal{N}')$, such that $r \xrightarrow{\lambda}{}^{*} r'$, with $r' \in QE(s')$. Use Proposition 4 to construct and reach r', and to show that $r' \in QE(s')$.

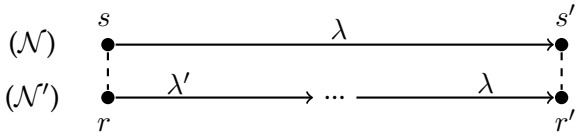
(c) $s \notin StableConf_Y$, for some $Y \in \mathcal{EC}_{\mathcal{N}}$, and the transition is justified by a simple edge wrt. Y



We show there exists $r' \in Conf(\mathcal{N}')$, such that $r \xrightarrow{0} r'$ and $r' \in QE(s')$.

- Note that since $s \notin StableConf_Y$ and $r \in QE(s)$, thus, resetter $\mathcal{R}_Y$ cannot be located at ℓ_{iniY} in configuration r, otherwise, there is a violation of Rule (6.0.2a) and Rule (6.0.2b).
- Propose $r' := r$ and $r \xrightarrow{0} r'$. We take r' as constructed and we show that $r' \in QE(s')$.
  - Values of integer variables existing in $\mathcal{N}$ are neither updated from s to s' (simple edges reset only clocks) nor from r to r' since $r' = r$. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
  - Both r' and s' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If r' and s' satisfy Rule (6.0.2a) then for the automaton $\mathcal{A}'_j$, for some $j \in \{1, \ldots, n\}$, (and its original version), $\ell_{r',j} = \ell_{s',j}$ and $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$, with $x \in \mathcal{X}(\mathcal{A}_j)$. Otherwise, r' and s' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$; the automaton $\mathcal{A}'_j$ is located in r' at the destination location of a transformed simple edge e wrt. Y, while the origin location of e is the current location of $\mathcal{A}_j$ in s'; $\nu_{r'}(rep(x)) = \nu_s(x)$, and $\nu_{r'}(t_x) = 0$, for each $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
  - Book-keeping variable $s_Y^{\mathcal{A}_i}$, with $1 \leq i \leq n$ and $Y \in \mathcal{EC}_\mathcal{N}$ is not updated from r to r', hence, has in r' value 1 if $\mathcal{A}'_i$ is located at a location which is the origin location of a transformed simple edge wrt. Y, otherwise value 0. Thus, r' satisfies Rule (6.2.3).
  - Variables $prio_Y$, $Y \in \mathcal{EC}_\mathcal{N}$, are not update from r to r', hence each of those has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus, r' satisfies (6.2.4).

(d) Transition is justified by a non-empty set of edges E, such that none of them is simple wrt. any $Y \in \mathcal{EC}_\mathcal{N}$.



We show there exists $r' \in Conf(\mathcal{N}')$, such that $r \xrightarrow{\lambda}{}^* r'$, with $r' \in QE(s')$. We distinguish the following cases based on the edges contained in E.

i. All edges in E are non-resetting edges.
   - Recall that the set of edges $E = \{(\ell_1, \alpha_1, \varphi_1, \vec{r}_1, \ell'_1), \ldots, (\ell_m, \alpha_m, \varphi_m, \vec{r}_m, \ell'_m)\}$ has a corresponding set of edges $\dot{P} = \{(\dot{\ell}_1, \dot{\alpha}_1, \dot{\varphi}_1, \dot{\vec{r}}_1, \dot{\ell}_1), \ldots, (\dot{\ell}_m, \dot{\alpha}_m, \dot{\varphi}_m, \dot{\vec{r}}_m, \dot{\ell}_m)\}$ which is obtained by applying Algorithm $\mathcal{K}$ on $\mathcal{N}$ (see transformation of non-resetting edges in Definition 27). Propose configuration $r' = (\vec{\ell}_r[\ell_{r,1} := \dot{\ell}'_1] \ldots [\ell_{r,m} := \dot{\ell}'_m], \nu_r[\dot{\vec{r}}_1] \ldots [\dot{\vec{r}}_m])$. We take r' as constructed and we show that $r' \in QE(s')$.

- Values of integer variables existing in $\mathcal{N}$ are updated in r' by taking edges in $\dot{P}$ if and only if those variables are updated in s' by taking edges in E. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
- Both configurations r' and s' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If r' and s' satisfy Rule (6.0.2a) then the automaton $\mathcal{A}'_j$, for some $j \in \{1, \ldots, n\}$, updated its current location in r' (by taking an edge $\dot{e}$ in $\dot{P}$) if and only if $\mathcal{A}_j$ updated its current location in s' (by taking the original version of $\dot{e}$ in E), thus, $\ell_{r',j} = \ell_{s',j}$ and because non-resetting edges do not update quasi-equal clocks, hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$, with $x \in \mathcal{X}(\mathcal{A}_j)$. Otherwise, r' and s' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$; the automaton $\mathcal{A}'_j$ is located in r' at the destination location of a transformed simple edge e wrt. Y, while the origin location of e is the current location of $\mathcal{A}_j$ in s'; $\nu_{r'}(rep(x)) = \nu_s(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
- The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$ and some $Y \in \mathcal{EC}_\mathcal{N}$, is set to 1 in r' if and only if the destination location of an edge of $\mathcal{A}'_i$ in $\dot{P}$ is the origin location of a transformed simple edge wrt. Y. If $\mathcal{A}'_i$ does not have an edge in $\dot{P}$ then the value of $s_Y^{\mathcal{A}_i}$ is 1 if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
- Variables $prio_Y$, $Y \in \mathcal{EC}_\mathcal{N}$, are not modified in r' since all edges in $\dot{P}$ are transformed non-resetting edges, which do not synchronise with edges of resetter $\mathcal{R}_Y$ that may update $prio_Y$. However, $prio_Y$ has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

The value of each clock $x \in \mathcal{X}(\mathcal{N})$ in s is equal to the value that $rep(x)$ and token t_x encode in r for x. Hence, $\nu_r \models \dot{\varphi}_1 \wedge \cdots \wedge \dot{\varphi}_m$, and $\nu_r[\vec{\dot{r}}_1] \ldots [\vec{\dot{r}}_m] \models I(\dot{\ell}'_1) \wedge \cdots \wedge I(\dot{\ell}'_m)$ because $I(\dot{\ell}'_1) = I(\Gamma(\ell'_1)), \ldots, I(\dot{\ell}'_m) = I(\Gamma(\ell'_m))$ by Function Γ (Definition 24); invariants of automata which do not change locations are still satisfied in r'. Hence, all edges in $\dot{P}$ are enabled. Thus, by the semantics of extended timed automata Definition (16), $r \xrightarrow{\tau} r$.

ii. There exists at least one complex edge wrt. Y in E.

- As a result of applying Algorithm $\mathcal{K}$ on $\mathcal{N}$, in $\mathcal{N}'$ transitions justified by taking transformed simple edges are prioritised over transitions justified by taking complex ones, therefore, use Proposition 4 to obtain and reach $\dot{r}$ where all transformed simple edges enabled in r have been taken. Note that $\dot{r} = r$ if there are no transformed simpled edges enabled in r.

- Recall that the set of edges $E = \{(\ell_1, \alpha_1, \varphi_1, \vec{r}_1, \ell'_1), \ldots, (\ell_m, \alpha_m, \varphi_m, \vec{r}_m, \ell'_m)\}$ has a corresponding set of edges $\dot{P} = \{(\dot{\ell}_1, \dot{\alpha}_1, \dot{\varphi}_1, \vec{\dot{r}}_1, \dot{\ell}_1), \ldots, (\dot{\ell}_m, \dot{\alpha}_m, \dot{\varphi}_m, \vec{\dot{r}}_m, \dot{\ell}_m)\}$ obtained by applying Algorithm $\mathcal{K}$ on $\mathcal{N}$ (see transformation of complex and non-resetting edges in Definition 27). Propose $r' = (\vec{\ell}_{\dot{r}}[\ell_{\dot{r},1} := \dot{\ell}'_1] \ldots [\ell_{\dot{r},m} := \dot{\ell}'_m], \nu_{\dot{r}}[\vec{\dot{r}}_1] \ldots [\vec{\dot{r}}_m])$. We take r' as constructed and we show that $r' \in QE(s')$.
  - Values of integer variables existing in $\mathcal{N}$ are updated in r' by taking edges in $\dot{P}$ if and only if those variables are updated in s' by taking edges in E. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
  - Both configurations r' and s' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If r' and s' satisfy Rule (6.0.2a) then the automaton $\mathcal{A}'_j$, for some $j \in \{1, \ldots, n\}$, updated its current location in r' (by taking an edge $\dot{E}$ in $\dot{P}$) if and only if $\mathcal{A}_j$ updated its current location in s' (by taking the original version of $\dot{e}$ in E), thus, $\ell_{r',j} = \ell_{s',j}$ and because complex edges do update quasi-equal clocks, value of t_x, with $x \in \mathcal{X}(\mathcal{A}_j)$, is set to 0 in r' if and only if clock x is reset in s', hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$. Otherwise, r' and s' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_{\mathcal{N}}$; the automaton $\mathcal{A}'_j$ is located in r' at the destination location of a transformed simple edge e wrt. Y, while the origin location of e is the current location of $\mathcal{A}_j$ in s'; $\nu_{r'}(rep(x)) = \nu_s(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
  - The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$ and some $Y \in \mathcal{EC}_{\mathcal{N}}$, is set to 1 in r' if and only if the destination location of an edge of $\mathcal{A}'_i$ in $\dot{P}$ is the origin location of a transformed simple edge wrt. Y. If $\mathcal{A}'_i$ does not have an edge in $\dot{P}$ then the value of $s_Y^{\mathcal{A}_i}$ is 1 if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
  - Variables $prio_Y$, $Y \in \mathcal{EC}_{\mathcal{N}}$, are not modified in r' since all edges in $\dot{P}$ are edges which do not synchronise with edges of resetter $\mathcal{R}_Y$ that update $prio_Y$. However, $prio_Y$ has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

  The value of each clock $x \in \mathcal{X}(\mathcal{N})$ in s is equal to the value that the representative $rep(x)$ and token t_x encode in $\dot{r}$ for x. Hence, $\nu_{\dot{r}} \models \dot{\varphi}_1 \wedge \cdots \wedge \dot{\varphi}_m$, and $\nu_{\dot{r}}[\vec{\dot{r}}_1] \ldots [\vec{\dot{r}}_m] \models I(\dot{\ell}'_1) \wedge \cdots \wedge I(\dot{\ell}'_m)$ because the invariants $I(\dot{\ell}'_1) = I(\Gamma(\ell'_1))$, $\ldots, I(\dot{\ell}'_m) = I(\Gamma(\ell'_m))$ by Function Γ (Definition 24); invariants of automata which do not change locations are still satisfied in r'. Hence, all edges in $\dot{P}$ are enabled. Thus, by the semantics of extended timed automata (Definition 16), $\dot{r} \xrightarrow{\tau} r'$.

- Let $S_1 = \{Y \in \mathcal{EC}_{\mathcal{N}} \mid r' \models \forall\, x \in Y \bullet \ell_{\mathcal{R}_Y} = \ell_{ini\mathcal{R}_Y} \wedge t_x = 0 \wedge prio_Y = 0\}$. Note that S_1 may be empty.
- Set $r := r'$ and execute the following algorithm while $S_1 \neq \emptyset$:

A. Pick $W \in S_1$, such that W has a higher index in $\mathcal{EC}^{prio}_{\mathcal{N}}$ than any other element in S_1.

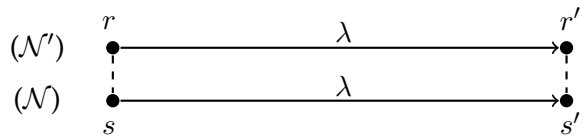
B. Consider the edge $e_W = (\ell_{ini\mathcal{R}_W}, u_W!, \varphi_W, \langle prio_W := 0\rangle, \ell_{nst\mathcal{R}_W}) \in E(\mathcal{R}_W)$ sending on the urgent broadcast channel u_W, (by Algorithm 27 there are no edges listening on u_W) and propose the configuration $r' = (\vec{\ell}_r[\ell_{\mathcal{R}_W} := \ell_{nst\mathcal{R}_W}], \nu_r[\langle prio_W := 0\rangle])$. It is easy to see that $r' \in QE(s')$.
Note that $\varphi_W = (\sum_{w\in W} t_w = 0)$. Recall that $W \in S_1$ if and only if $r \models \forall\, x \in W \bullet \ell_{\mathcal{R}_W} = \ell_{ini\mathcal{R}_W} \wedge t_x = 0 \wedge prio_W = 0$. Thus $\nu_r \models \varphi_W$. Note as well that $\nu_r[\langle prio_W := 0\rangle] \models I(\ell_{nst\mathcal{R}_W})$ because $I(\ell_{nst\mathcal{R}_W})$ is the invariant of an urgent location constraining the lapse of time with the help of a fresh clock, and if invariants of other locations are satisfied in r then they are still satisfied in r' since only $\mathcal{R}_W$ changes location in r'. Hence, e_W is enabled.
Recall the edge $e = (\ell_{ini\mathcal{R}_W}, reset_W?, \varphi, \vec{r}, \ell_{nst\mathcal{R}_W}) \in E(\mathcal{R}_W)$ which is not enabled because there are no transformed simple edges sending on $reset_W$ enabled at the current time, which is deduced from $r \models \forall\, x \in W \bullet \ell_{\mathcal{R}_W} = \ell_{ini\mathcal{R}_W} \wedge t_x = 0 \wedge prio_W = 0$.
Since W has a higher index in $\mathcal{EC}^{prio}_{\mathcal{N}}$ than any other element in S_1, thus, in $\mathcal{N}'$ the transition justified by e_W has the highest priority at the current time (see the prioritisation mechanism implemented by Algorithm $\mathcal{K}$, Definition 27). Hence, $r \xrightarrow{u_W} r'$.

C. Set $S_1 := S_1 \setminus \{W\}$, $r := r'$.

Note that the above algorithm terminates because we leave its loop by decrementing the size of S_1 in each iteration until $S_1 = \emptyset$.

5. $\forall\, CF \in \mathcal{CF}_{\mathcal{N}} \forall\, r \in QE(s) \bullet r \xrightarrow{\lambda} r' \wedge r' \models_{\delta'} \Omega_0(CF) \implies \exists\, s \xrightarrow{\lambda}{}^{*}\, s' \bullet r' \in QE(s')$. We distinguish the transition $r \xrightarrow{\lambda} r'$ as follows:

(a) transition is justified by a delay $\lambda = d \geq 0$.

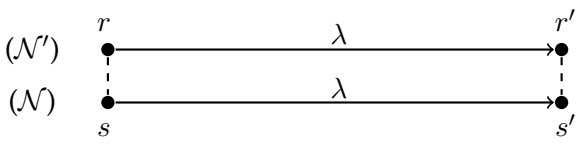


We show there exists $s' \in Conf(\mathcal{N})$, such that $s \xrightarrow{\lambda} s'$, with $s' \in QE(r')$. We distinguish two cases based on d, namely, $d = 0$ and $d > 0$. The first case $d = 0$ is trivial. Since this transition in $\mathcal{N}'$ is of the type $\langle \vec{\ell}_r, \nu_r\rangle \xrightarrow{0} \langle \vec{\ell}_r, \nu_r + 0\rangle$, thus, $r = r'$, and we propose s', such that $s = s'$.

Hence, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{0} s'$. Trivially, $s' \in QE(r')$. We show the second case $d > 0$.

- This transition in $\mathcal{N}'$ is of the type $\langle \vec{\ell}_r, \nu_r \rangle \xrightarrow{d} \langle \vec{\ell}_r, \nu_r + d \rangle$. Each resetter is located in r at its initial location, i.e. $\ell_{r,\mathcal{R}_Y} = \ell_{ini\mathcal{R}_Y}$, for all $Y \in \mathcal{EC}_\mathcal{N}$ (all other locations of each resetter are urgent, thus, delays $d > 0$ are not possible at those locations).
- Since $r \in QE(s)$, we know that $s \in \bigcap_{Y \in \mathcal{EC}_\mathcal{N}} StableConf_Y$, otherwise, if $s \notin StableConf_Y$, for some $Y \in \mathcal{EC}_\mathcal{N}$, then quasi-equality would not hold for Y (see Lemma 1) after the delay d.
- Propose $s' := \langle \vec{\ell}_{s'}, \nu_{s'} + d \rangle$, where all clocks in $\mathcal{X}(\mathcal{N})$ have advanced d time units. We take s' as constructed and we show that $r' \in QE(s')$.
  - Values of integer variables existing in $\mathcal{N}$ are not updated in delay transitions. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
  - Configurations r' and s' do not satisfy Rule (6.0.2b), since in r' each resetter is located at its initial location. Configurations r' and s' do satisfy Rule (6.0.2a), therefore, the location of $\mathcal{A}_j$, with $j \in \{1, \ldots, n\}$, is the same in s' as the location of $\mathcal{A}'_j$ in r', i.e. $\ell_{s',j} = \ell_{r',j}$. The value of each $x \in \mathcal{X}(\mathcal{A}_j)$ in s' is equal to the value that the representative $rep(x)$ and the token t_x encode in r' for x, thus, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$.
  - The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$ and some $Y \in \mathcal{EC}_\mathcal{N}$, is 1 in r' if and only if $\mathcal{A}'_i$ is located at the origin location of a transformed simple edge wrt. Y. Thus, r' satisfies Rule (6.2.3).
  - All resetters are now located at their initial locations and their respective variables $prio_Y$, with $Y \in \mathcal{EC}_\mathcal{N}$, have value 0. Thus r' satisfies (6.2.4).
- Note that the invariants of the current locations of automata in s are invariants equivalent to the transformed ones of current locations of automata in r, i.e. $I(\ell_{r,i}) = \Gamma(I(\ell_{s,i}))$, for each $1 \leq i \leq n$, which have been introduced by Algorithm 27. Hence, $\nu_s + d \models \bigwedge_{i=1}^{n} I(\ell_{s',i})$. Thus by the semantics of extended timed automata (Definition 16), $s \xrightarrow{d} s'$.

(b) transition is justified by a non-empty set of edges E.



We show there exists $s' \in Conf(\mathcal{N})$, such that $s \xrightarrow{\lambda}{}^* s'$, $r' \in QE(s')$.

In the following we distinguish five cases based on the edges contained in E. We want to point out that each case is justified by the set E, that is, each edge in that set is taken to justify the underlying transition. Therefore, some combinations of edges are not possible to be contained in E. For instance, transformed simple edges and transformed complex ones cannot be contained in E, since that would imply to take all those edges in one transition, namely, a broadcast one, however, neither simple edges nor their transformed version are equipped with a mechanism to synchronise with complex edges and transformed complex ones, respectively. We refer the interested reader to Definition 22 and Algorithm $\mathcal{K}$, Definition 27 to respectively recall the definition of resetting edges and their transformation.

i. The set E contains edges of the kind: (1) transformed complex edges and (or), (2) transformed non-resetting edges.
   - Recall that the set of edges $E = \{e_1 = (\ell_1, \alpha_1, \varphi_1, \vec{r}_1, \ell'_1), \ldots, e_n = (\ell_n, \alpha_n, \varphi_n, \vec{r}_n, \ell'_n)\}$ obtained by applying Algorithm $\mathcal{K}$ (Definition 27) on $\mathcal{N}$, has a corresponding set of original edges $\dot{P} = \{\dot{e}_1 = (\dot{\ell}_1, \dot{\alpha}_1, \dot{\varphi}_1, \dot{\vec{r}}_1, \dot{\ell}_1), \ldots, \dot{e}_n = (\dot{\ell}_n, \dot{\alpha}_n, \dot{\varphi}_n, \dot{\vec{r}}_n, \dot{\ell}_n)\}$.
     Propose $s' = (\vec{\ell}_s[\ell_{s,1} := \dot{\ell}'_1] \ldots [\ell_{s,n} := \dot{\ell}'_n], \nu_s[\dot{\vec{r}}_1] \ldots [\dot{\vec{r}}_n])$. We take s' as constructed and we show that $r' \in QE(s')$.
     - Values of integer variables existing in $\mathcal{N}$ are updated in s' by taking edges in $\dot{P}$ if and only if those variables are updated in r' by taking edges in E. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
     - Both configuration r' and s' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If r' and s' satisfy Rule (6.0.2a) then the automaton $\mathcal{A}_j$, for some $j \in \{1, \ldots, n\}$, updated its current location in s' (by taking an edge $\dot{e}$ in $\dot{P}$) if and only if $\mathcal{A}'_j$ updated its current location in r' (by taking the transformed version of $\dot{e}$ in E), thus, $\ell_{r',j} = \ell_{s',j}$, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$, with $x \in \mathcal{X}(\mathcal{A}_j)$. Otherwise, r' and s' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$; the automaton $\mathcal{A}_j$ is located in s' at the origin location of a simple edge e wrt. Y, while the destination location of e is the current location of $\mathcal{A}'_j$ in r'; $\nu_{r'}(rep(x)) = \nu_s(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
     - The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$ and some $Y \in \mathcal{EC}_\mathcal{N}$, is set to 1 in r' if and only if the destination location of an edge of $\mathcal{A}'_i$ in E is the origin location of a transformed simple edge wrt. Y. If $\mathcal{A}'_i$ does not have an edge in E then the value of $s_Y^{\mathcal{A}_i}$ is 1 if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
     - Variables $prio_Y$, $Y \in \mathcal{EC}_\mathcal{N}$, are not modified in r' since no edge in E synchronises with an edge of resetter $\mathcal{R}_Y$

that may update $prio_Y$. However, $prio_Y$ has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

The value of each clock $x \in \mathcal{X}(\mathcal{N})$ in s is encoded by the representative $rep(x)$ and token t_x in r. Moreover, recall that the guards $\varphi_1, \ldots, \varphi_n$ of edges in E are obtained by Algorithm $\mathcal{K}$ from guards $\dot{\varphi}_1, \ldots, \dot{\varphi}_n$ of edges in $\dot{P}$. Hence, $\nu_s \models \dot{\varphi}_1 \wedge \cdots \wedge \dot{\varphi}_n$, and $\nu_s[\vec{\dot{r}}_1] \ldots [\vec{\dot{r}}_n] \models I(\dot{\ell}'_1) \wedge \cdots \wedge I(\dot{\ell}'_n)$ since the invariants $I(\ell_{r',i}) = \Gamma(I(\ell_{s',i}))$, for each $1 \leq i \leq n$, which have been introduced by Algorithm 27. Hence, all edges in $\dot{P}$ are enabled. Thus, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{\tau} s'$.

ii. The set E consists exclusively of the edge $e = (\ell_{ini\mathcal{R}_Y}, u_Y!, \varphi, \langle prio_Y := 1\rangle, \ell_{nst\mathcal{R}_Y}) \in E(\mathcal{R}_Y)$, for some $Y \in \mathcal{EC}_\mathcal{N}$.

- Recall that e has an output on the urgent broadcast channel u_Y, and by Algorithm $\mathcal{K}$ there is no edge in $\mathcal{N}'$ with an input on u_Y.
- Propose $s' := s$. We take s' as constructed and we show that $r' \in QE(s')$.
  - Values of integer variables existing in $\mathcal{N}$ are neither updated in r' (by e) nor in s'. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
  - Both configurations r' and s' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If r' and s' satisfy Rule (6.0.2a) then for the automaton $\mathcal{A}_j$ (and for its transformed version), for some $j \in \{1, \ldots, n\}$, $\ell_{r',j} = \ell_{s',j}$ and because e does not update clocks, hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$, with $x \in \mathcal{X}(\mathcal{A}_j)$.
    Otherwise, r' and s' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$; the automaton $\mathcal{A}_j$ is located in s' at the origin location of a simple edge e wrt. Y, while the destination location of e is the current location of $\mathcal{A}'_j$ in r'; $\nu_{r'}(rep(x)) = \nu_s(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
  - The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$ and some $Y \in \mathcal{EC}_\mathcal{N}$, is 1 in r' if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
  - The variable $prio_Y$, with $Y \in \mathcal{EC}_\mathcal{N}$, has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

  We propose a 0-delay transition. Hence, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{0} s'$.

iii. The set E exclusively consists of edges with an input or output on the channel $return$.

- Recall that inputs and outputs on the channel $return$ occur only in edges of resetters of $\mathcal{N}'$ (see construction of resetters by Algorithm $\mathcal{K}$, Definition 27). Hence, E exclusively consists of edges of resetters.
- Let $P := \{(\ell, \alpha, \varphi, \vec{r}, \ell') \in SimpEdges_Y(\mathcal{N}) \mid \exists i \in \mathbb{N} \bullet \ell_{s,i} = \ell \wedge \nu_s \models \varphi \wedge \nu_s[\vec{r}\,] \models I(\ell') \wedge Y \in \mathcal{EC}_\mathcal{N}\}$, i.e. the set of simple edges enabled in the configuration s.
- We distinguish two cases based on the set P. If $P = \emptyset$ then:
  - Propose $s' := s$. We take s' as constructed and we show that $r' \in QE(s')$.
    - Values of integer variables existing in $\mathcal{N}$ are neither updated in r' nor in s'. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
    - Both configurations r' and s' do satisfy Rule (6.0.2a) then automaton $\mathcal{A}_j$, for each $j \in \{1, \ldots, n\}$, and $\mathcal{A}'_j$ are located at the same location, i.e. $\ell_{r',j} = \ell_{s',j}$, and because some edges in E do update $rep(x)$ and t_x, for some $x \in \mathcal{X}(\mathcal{A}_j)$, hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$.
    - The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$, for some $Y \in \mathcal{EC}_\mathcal{N}$, is 1 in r' if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
    - The variable $prio_Y$, with $Y \in \mathcal{EC}_\mathcal{N}$, has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).
  - We propose a 0-delay transition. Hence, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{0} s'$.

  Otherwise, $P \neq \emptyset$ and we execute the following algorithm while $P \neq \emptyset$:
  - For some $1 \leq i \leq n$, pick the following simple edge $e = (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in P$, such that $e \in E(\mathcal{A}_i)$.
  - Propose $s' := (\vec{\ell}_s[\ell_{s,i} := \ell'_1], \nu_s[\langle x := 0\rangle])$. Recall that e is an edge enabled in s. This implies that $\nu_s \models \varphi \wedge \nu_s[\langle x := 0\rangle] \models I(\ell')$. Thus, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{\tau} s'$.
    Recall that guards of simple edges consist of a single clock constraint $y \geq c$, with $y \in Y$, $Y \in \mathcal{EC}_\mathcal{N}$ and $c \in \mathbb{N}^{>0}$. Note that the above transition only updates the value of clock x in s'. Therefore, this update does not hinder other enabled simple edges in s' to be taken.
  - Set $P := P \setminus \{e\}$ and $s := s'$.

  Note that the above algorithm terminates because we leave its loop by decrementing the size of P in each iteration until $P = \emptyset$. We take s' as constructed and we show that $r' \in QE(s')$.

- Values of integer variables existing in $\mathcal{N}$ are neither updated in r' nor in s'. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
- Both configurations r' and s' satisfy Rule (6.0.2a) then automata $\mathcal{A}_j$, for each $j \in \{1, \ldots, n\}$, and $\mathcal{A}'_j$ are located at the same location, i.e. $\ell_{r',j} = \ell_{s',j}$, and because some edges in E do update $rep(x)$ and t_x, for some $x \in \mathcal{X}(\mathcal{A}_j)$, hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$.
- The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$, for some $Y \in \mathcal{EC}_{\mathcal{N}}$, is 1 in r' if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
- The variable $prio_Y$, with $Y \in \mathcal{EC}_{\mathcal{N}}$, has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

iv. The set E exclusively consists of transformed simple edges.

- Recall that $r' \models_{\delta'} \Omega_0(CF)$. Let CV be the set of logical variables introduced by Ω in transformations for locations and clock constraints occurring in CF.
  Let $Cks := \{x \in \mathcal{X}(\mathcal{N}) \mid \tilde{x} \in CV \wedge \delta'(\tilde{x}) = 1\}$. Let $\dot{E} = \{(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_i) \mid 1 \leq i \leq n \wedge x \in Cks \wedge Y \in \mathcal{EC}_{\mathcal{N}} \wedge \ell_{s,i} = \ell\}$.
  Note that any edge $\dot{e} = (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell')$ of $\dot{E}$ is enabled in s, since its transformed version is enabled in r and was taken in the transition from r to r'. This information is obtained from the logical variables occurring in δ' with value true.
- If $\dot{E} = \emptyset$ then propose $s' := s$. We propose a 0-delay transition. Hence, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{0} s'$. Take s' as constructed to show $r' \in QE(s')$:
  - In 0-delays no integer variables are updated. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
  - Both configurations s' and r' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If s' and r' satisfy Rule (6.0.2a) then the automata $\mathcal{A}_j$, for some $j \in \{1, \ldots, n\}$, and $\mathcal{A}'_j$ are located at the same location, i.e. $\ell_{r',j} = \ell_{s',j}$, and the value of the clock $x \in \mathcal{X}(\mathcal{A}_j)$ in s' is encoded by $rep(x)$ and t_x in r', hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$.
    Otherwise, s' and r' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_{\mathcal{N}}$, but at $\ell_{nst\mathcal{R}_Y}$; the automaton $\mathcal{A}_j$ is located in s' at the origin location of a simple edge e wrt. Y, while the destination location of e is the current location of $\mathcal{A}'_j$ in r'; $\nu_{r'}(rep(x)) = \nu_{s'}(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.

– The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$, for some $Y \in \mathcal{EC}_\mathcal{N}$, is 1 in r' if and only if the destination location of an edge of $\mathcal{A}'_i$ in E is the origin location of a transformed simple edge wrt. Y. If $\mathcal{A}'_i$ does not have an edge in E then the value of $s_Y^{\mathcal{A}_i}$ is 1 if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
– Variable $prio_Y$, $Y \in \mathcal{EC}_\mathcal{N}$, has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

Otherwise, $\dot{E} \neq \emptyset$ and we execute the following steps until $\dot{E} = \emptyset$:

– For some $1 \leq i \leq n$ and some $Y \in \mathcal{EC}_\mathcal{N}$, pick an edge $\dot{e} = (\dot{\ell}, \dot{\alpha}, \dot{\varphi}, \dot{\vec{r}}, \dot{\ell}') \in \dot{E}$, such that $\dot{e} \in SimpEdges_Y(\mathcal{A}_i)$.
– Propose $s' = (\vec{\ell}_s[\ell_{s,i} := \dot{\ell}'_i], \nu_s[\dot{\vec{r}}_i])$. Take s' as constructed, we show that $r' \in QE(s')$ holds:
  * Simple edges do not update integer variables of $\mathcal{N}$. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
  * Both s' and r' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If s' and r' satisfy Rule (6.0.2a) then the automaton $\mathcal{A}_j$, for some $j \in \{1, \ldots, n\}$, updated its current location in s' (by taking an edge $\dot{e}$ in $\dot{E}$) if and only if $\mathcal{A}'_j$ updated its current location in r' (by taking the transformed version of $\dot{e}$ in E), thus, $\ell_{r',j} = \ell_{s',j}$ and because simple edges do reset quasi-equal clocks, hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$, with $x \in \mathcal{X}(\mathcal{A}_j)$.
    Otherwise, s' and r' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$, but at $\ell_{nst\mathcal{R}_Y}$; the automaton $\mathcal{A}_j$ is located in s' at the origin location of a simple edge e in $\dot{E}$, while the destination location of e is the current location of $\mathcal{A}'_j$ in r'; $\nu_{r'}(rep(x)) = \nu_{s'}(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
  * The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$, for some $Y \in \mathcal{EC}_\mathcal{N}$, is set to 1 in r' if and only if the destination location of an edge of $\mathcal{A}'_i$ in E is the origin location of a transformed simple edge wrt. Y. If $\mathcal{A}'_i$ does not have an edge in E then the value of $s_Y^{\mathcal{A}_i}$ is 1 if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
  * Variable $prio_Y$, $Y \in \mathcal{EC}_\mathcal{N}$, has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

  The value in r that the representative $rep(x)$ and token t_x, with $Y \in \mathcal{EC}_\mathcal{N}$, encode in r is equal to the value of

x in s. Hence, $\nu_s \models \dot{\varphi}$, and $\nu_s[\dot{\vec{r}}] \models I(\dot{\ell}')$ since the invariant of the current location of automaton $\mathcal{A}_i$ in s', is an invariant equivalent (by Function Γ) to the transformed invariant of the current location of automaton $\mathcal{A}'_i$ in r'. Hence, edge $\dot{e}$ is enabled. Thus, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{\tau} s'$.

Note that the above transition only updates the value of clock x in s'. Therefore, this update does not hinder other enabled simple edges in s' to be taken.

– Eliminate $\dot{e}$ from $\dot{E}$, and set $s := s'$.

Note that the above algorithm terminates because we leave its loop by decrementing the size of $\dot{E}$ in each iteration until $\dot{E} = \emptyset$.

v. The set E consists exclusively of the edge $e = (\ell_{nst\mathcal{R}_Y}, \tau, \varphi, \langle prio_Y := 0\rangle, Tlock_Y) \in E(\mathcal{R}_Y)$, for some $Y \in \mathcal{EC}_\mathcal{N}$.

- Recall that $r' \models_{\delta'} \Omega_0(CF)$. Let CV be the set of logical variables introduced by Ω in transformations for locations and clock constraints occurring in CF.
  Let $Cks := \{x \in \mathcal{X}(\mathcal{N}) \mid \tilde{x} \in CV \wedge \delta'(\tilde{x}) = 1\}$. Let $\dot{E} = \{(\ell, \alpha, \varphi, \langle x := 0\rangle, \ell') \in SimpEdges_Y(\mathcal{A}_i) \mid 1 \leq i \leq n \wedge x \in Cks \wedge \ell_{s,i} = \ell\}$.
  Note that any edge $\dot{e} = (\ell, \alpha, \varphi, \langle x := 0\rangle, \ell')$ of $\dot{E}$ is enabled in s, since its transformed version is enabled in r and was taken in the transition from r to r'. This information is obtained from the logical variables occurring in δ' with value true.
- If $\dot{E} = \emptyset$ then propose $s' := s$. We propose a 0-delay transition. Hence, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{0} s'$. Take s' as constructed to show $r' \in QE(s')$:
  – In 0-delays no integer variables are updated. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
  – Both configurations s' and r' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If s' and r' satisfy Rule (6.0.2a) then the automata $\mathcal{A}_j$, for some $j \in \{1, \ldots, n\}$, and $\mathcal{A}'_j$ are located at the same location, i.e. $\ell_{r',j} = \ell_{s',j}$, and the value of the clock $x \in \mathcal{X}(\mathcal{A}_j)$ in s' is encoded by $rep(x)$ and t_x in r', hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$.
    Otherwise, s' and r' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$, but at $\ell_{nst\mathcal{R}_Y}$; the automaton $\mathcal{A}_j$ is located in s' at the origin location of a simple edge e wrt. Y, while the destination location of e is the current location of $\mathcal{A}'_j$ in r'; $\nu_{r'}(rep(x)) = \nu_{s'}(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.

– The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$, for some $Y \in \mathcal{EC}_\mathcal{N}$, is 1 in r' if and only if the destination location of an edge of $\mathcal{A}'_i$ in E is the origin location of a transformed simple edge wrt. Y. If $\mathcal{A}'_i$ does not have an edge in E then the value of $s_Y^{\mathcal{A}_i}$ is 1 if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
– Variable $prio_Y$, $Y \in \mathcal{EC}_\mathcal{N}$, has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

Otherwise, $\dot{E} \neq \emptyset$ and we execute the following steps until $\dot{E} = \emptyset$:

– For some $1 \leq i \leq n$ and some $Y \in \mathcal{EC}_\mathcal{N}$, pick an edge $\dot{e} = (\dot{\ell}, \dot{\alpha}, \dot{\varphi}, \dot{\vec{r}}, \dot{\ell}') \in \dot{E}$, such that $\dot{e} \in SimpEdges_Y(\mathcal{A}_i)$.
– Propose $s' = (\vec{\ell}_s[\ell_{s,i} := \dot{\ell}'_i], \nu_s[\dot{\vec{r}}_i])$. Take s' as constructed, we show that $r' \in QE(s')$ holds:
  * Simple edges do not update integer variables of $\mathcal{N}$. Hence, for each $v \in V(\mathcal{N})$ the value of v is the same in s' and in r', i.e. $\nu_{s'}(v) = \nu_{r'}(v)$. Thus, r' and s' satisfy Rule (6.2.1).
  * Both s' and r' satisfy either Rule (6.0.2a) or Rule (6.0.2b). If s' and r' satisfy Rule (6.0.2a) then the automaton $\mathcal{A}_j$, for some $j \in \{1, \ldots, n\}$, updated its current location in s' (by taking an edge $\dot{e}$ in $\dot{E}$) if and only if $\mathcal{A}'_j$ updated its current location in r' (by taking the transformed version of $\dot{e}$ in E), thus, $\ell_{r',j} = \ell_{s',j}$ and because simple edges do reset quasi-equal clocks, hence, $\nu_{s'}(x) = \nu_{r'}(rep(x)) \cdot \nu_{r'}(t_x)$, with $x \in \mathcal{X}(\mathcal{A}_j)$.
    Otherwise, s' and r' satisfy Rule (6.0.2b) and the resetter $\mathcal{R}_Y$ is not located at $\ell_{ini\mathcal{R}_Y}$, $Y \in \mathcal{EC}_\mathcal{N}$, but at $\ell_{nst\mathcal{R}_Y}$; the automaton $\mathcal{A}_j$ is located in s' at the origin location of a simple edge e in $\dot{E}$, while the destination location of e is the current location of $\mathcal{A}'_j$ in r'; $\nu_{r'}(rep(x)) = \nu_{s'}(x)$, and $\nu_{r'}(t_x) = 0$, for each clock $x \in \mathcal{X}(\mathcal{A}_j)$ reset by e.
  * The value of the book-keeping variable $s_Y^{\mathcal{A}_i}$, for some index $i \in \{1, \ldots, n\}$, for some $Y \in \mathcal{EC}_\mathcal{N}$, is set to 1 in r' if and only if the destination location of an edge of $\mathcal{A}'_i$ in E is the origin location of a transformed simple edge wrt. Y. If $\mathcal{A}'_i$ does not have an edge in E then the value of $s_Y^{\mathcal{A}_i}$ is 1 if $\mathcal{A}'_i$ is located in r' at the origin location of a transformed simple edge wrt. Y, otherwise 0. Thus, r' satisfies Rule (6.2.3).
  * Variable $prio_Y$, $Y \in \mathcal{EC}_\mathcal{N}$, has value 1 if the resetter $\mathcal{R}_Y$ is located at $\ell_{nst\mathcal{R}_Y}$ in r', otherwise 0. Thus r' satisfies (6.2.4).

  The value in r that the representative $rep(x)$ and token t_x, with $Y \in \mathcal{EC}_\mathcal{N}$, encode in r is equal to the value of

x in s. Hence, $\nu_s \models \dot{\varphi}$, and $\nu_s[\vec{\dot{r}}] \models I(\dot{\ell}')$ since the invariant of the current location of automaton $\mathcal{A}_i$ in s', is an invariant equivalent (by Function Γ) to the transformed invariant of the current location of automaton $\mathcal{A}'_i$ in r'. Hence, edge $\dot{e}$ is enabled. Thus, by the semantics of extended timed automata (Definition 16), $s \xrightarrow{\tau} s'$.

Note that the above transition only updates the value of clock x in s'. Therefore, this update does not hinder other enabled simple edges in s' to be taken.

– Eliminate $\dot{e}$ from $\dot{E}$, and set $s := s'$.

Note that the above algorithm terminates because we leave its loop by decrementing the size of $\dot{E}$ in each iteration until $\dot{E} = \emptyset$.

□

Appendix B

Detailed Verification Results

B.1 Introduction

This appendix complements Chapter 8 and contains tables with detailed verification results for the case studies: *FS*, *CD*, *LS*, *EP*, *TT*, *PG*, *FB*.

These verification results are provided for networks transformed with Algorithm $\mathcal{K}^{\triangledown}$ on the output of Algorithm $\mathcal{K}$ (the broadcast version).

Note that in each case study we report on the minimal instances that a network consists of, and on the maximal instances that by our experience are possible to model check in our experimental environment.

B.2 Tables

Network	C	kStates	M	$t(s)$	Network	C	kStates	M	$t(s)$
LS-1	8	0.44	6.0	0.0	*LS-1*$\mathcal{K}^{\triangledown}$	4	0.91	6.5	0.0
LS-2	10	1.02	6.1	0.0	*LS-2*$\mathcal{K}^{\triangledown}$	4	2.17	6.7	0.0
LS-3	12	3.02	6.3	0.0	*LS-3*$\mathcal{K}^{\triangledown}$	4	5.50	7.1	0.0
LS-4	14	10.26	6.9	0.2	*LS-4*$\mathcal{K}^{\triangledown}$	4	15.44	8.0	0.2
LS-5	16	37.82	9.2	1.3	*LS-5*$\mathcal{K}^{\triangledown}$	4	48.37	11.3	0.9
LS-6	18	145.14	18.4	6.3	*LS-6*$\mathcal{K}^{\triangledown}$	4	166.45	24.3	4.1
LS-7	20	568.70	56.5	30.7	*LS-7*$\mathcal{K}^{\triangledown}$	4	611.50	76.2	18.1
LS-8	22	2,251.38	224.2	150.1	*LS-8*$\mathcal{K}^{\triangledown}$	4	2,337.20	284.7	82.4
LS-9	24	8,759.10	876.5	723.6	*LS-9*$\mathcal{K}^{\triangledown}$	4	8,822.47	1,387.2	352.7
LS-10	26	33,743.86	3,401.2	3,427.3	*LS-10*$\mathcal{K}^{\triangledown}$	4	34,451.53	5,556.2	1,561.8

TABLE B.1: Detailed results for the case study LS. Row X-$N(\mathcal{K}^{\triangledown})$ gives the figures for benchmark X with N components (and $\mathcal{K}^{\triangledown}$ applied). 'C' gives the number of clocks in the network, 'kStates' the number of 10^3 visited states, 'M' memory usage in MB, and '$t(s)$' verification time in secs.

Network	C	kStates	M	$t(s)$	Network	C	kStates	M	$t(s)$
CD-1	2	0.01	5.8	0.0	*CD-1*$\mathcal{K}^{\nabla}$	1	0.01	6.0	0.0
CD-2	3	0.04	5.8	0.0	*CD-2*$\mathcal{K}^{\nabla}$	1	0.03	6.2	0.0
CD-3	4	0.14	5.9	0.0	*CD-3*$\mathcal{K}^{\nabla}$	1	0.08	6.4	0.0
CD-4	5	0.50	6.0	0.0	*CD-4*$\mathcal{K}^{\nabla}$	1	0.16	6.7	0.0
CD-5	6	1.74	6.2	0.0	*CD-5*$\mathcal{K}^{\nabla}$	1	0.37	6.9	0.0
CD-6	7	6.07	6.7	0.1	*CD-6*$\mathcal{K}^{\nabla}$	1	0.82	7.2	0.0
CD-7	8	20.92	8.4	0.4	*CD-7*$\mathcal{K}^{\nabla}$	1	1.86	7.5	0.0
CD-8	9	71.07	14.0	1.8	*CD-8*$\mathcal{K}^{\nabla}$	1	4.17	8.0	0.1
CD-9	10	238.57	31.3	7.4	*CD-9*$\mathcal{K}^{\nabla}$	1	9.30	9.0	0.2
CD-10	11	792.52	90.3	29.3	*CD-10*$\mathcal{K}^{\nabla}$	1	20.57	10.8	0.5
CD-11	12	2,609.51	286.0	113.8	*CD-11*$\mathcal{K}^{\nabla}$	1	45.16	14.3	1.2
CD-12	13	8,527.73	959.8	453.0	*CD-12*$\mathcal{K}^{\nabla}$	1	98.41	21.7	3.1
CD-13	14	27,688.28	2,920.5	1,719.2	*CD-13*$\mathcal{K}^{\nabla}$	1	213.11	38.6	7.5
CD-14	15	89,396.89	9,492.4	6,457.8	*CD-14*$\mathcal{K}^{\nabla}$	1	458.88	73.4	18.1
CD-15	16	–	–	–	*CD-15*$\mathcal{K}^{\nabla}$	1	983.18	146.8	42.9

TABLE B.2: Detailed results for the case study CD.

Network	C	States	M	$t(s)$	Network	C	States	M	$t(s)$
FB-1	3	39	5.7	0.0	*FB-1*$\mathcal{K}^{\nabla}$	3	71	5.9	0.0
FB-2	4	51	5.8	0.0	*FB-2*$\mathcal{K}^{\nabla}$	3	83	5.9	0.0
FB-3	5	75	5.8	0.0	*FB-3*$\mathcal{K}^{\nabla}$	3	107	6.0	0.0
FB-4	6	123	5.8	0.0	*FB-4*$\mathcal{K}^{\nabla}$	3	155	6.1	0.0
FB-5	7	219	5.9	0.0	*FB-5*$\mathcal{K}^{\nabla}$	3	251	6.2	0.0
FB-6	8	411	6.0	0.0	*FB-6*$\mathcal{K}^{\nabla}$	3	443	6.4	0.0
FB-7	9	795	6.1	0.0	*FB-7*$\mathcal{K}^{\nabla}$	3	827	6.5	0.0
FB-8	10	1,563	6.3	0.0	*FB-8*$\mathcal{K}^{\nabla}$	3	1,595	6.7	0.0
FB-9	11	3,099	6.6	0.1	*FB-9*$\mathcal{K}^{\nabla}$	3	3,131	7.0	0.1
FB-10	12	6,171	7.2	0.4	*FB-10*$\mathcal{K}^{\nabla}$	3	6,203	7.6	0.2
FB-11	13	12,315	8.3	0.8	*FB-11*$\mathcal{K}^{\nabla}$	3	12,347	8.6	0.5
FB-12	14	24,603	10.5	9.5	*FB-12*$\mathcal{K}^{\nabla}$	3	24,635	10.6	1.3
FB-13	15	49,179	14.8	80.3	*FB-13*$\mathcal{K}^{\nabla}$	3	49,211	15.0	2.9
FB-14	16	98,331	25.2	424.7	*FB-14*$\mathcal{K}^{\nabla}$	3	98,363	23.7	6.6
FB-15	17	196,635	45.6	2,100.6	*FB-15*$\mathcal{K}^{\nabla}$	3	196,667	40.9	15.0
FB-16	18	393,243	86.7	9,427.0	*FB-16*$\mathcal{K}^{\nabla}$	3	393,275	76.7	33.3
FB-17	19	–	–	–	*FB-17*$\mathcal{K}^{\nabla}$	3	786,491	146.5	76.5
					FB-18$\mathcal{K}^{\nabla}$	3	1,572,923	286.0	172.0
					FB-19$\mathcal{K}^{\nabla}$	3	3,145,787	564.7	384.6
					FB-20$\mathcal{K}^{\nabla}$	3	6,291,515	1,122.0	839.0
					FB-21$\mathcal{K}^{\nabla}$	3	12,582,971	2,318.9	1,851.5
					FB-22$\mathcal{K}^{\nabla}$	3	25,165,883	4,631.4	3,997.3
					FB-23$\mathcal{K}^{\nabla}$	3	50,331,707	8,732.3	8,604.0

TABLE B.3: Detailed results for the case study FB.

Network	C	States	M	$t(s)$	Network	C	States	M	$t(s)$
EP-1	2	15	6.0	0.0	*EP-1*$\mathcal{K}^{\nabla}$	1	20	6.4	0.0
EP-2	3	24	6.1	0.0	*EP-2*$\mathcal{K}^{\nabla}$	1	39	6.7	0.0
EP-3	4	39	6.2	0.0	*EP-3*$\mathcal{K}^{\nabla}$	1	44	7.0	0.0
EP-4	5	66	6.2	0.0	*EP-4*$\mathcal{K}^{\nabla}$	1	71	7.3	0.0
EP-5	6	117	6.4	0.0	*EP-5*$\mathcal{K}^{\nabla}$	1	122	7.6	0.0
EP-6	7	216	6.5	0.0	*EP-6*$\mathcal{K}^{\nabla}$	1	221	8.0	0.0
EP-7	8	411	6.6	0.0	*EP-7*$\mathcal{K}^{\nabla}$	1	416	8.4	0.0
EP-8	9	798	6.7	0.0	*EP-8*$\mathcal{K}^{\nabla}$	1	803	8.8	0.0
EP-9	10	1,569	7.0	0.0	*EP-9*$\mathcal{K}^{\nabla}$	1	1,574	9.2	0.1
EP-10	11	3,108	7.3	0.1	*EP-10*$\mathcal{K}^{\nabla}$	1	3,113	9.9	0.1
EP-11	12	6,183	7.9	0.2	*EP-11*$\mathcal{K}^{\nabla}$	1	6,188	10.8	0.3
EP-12	13	12,330	8.9	0.5	*EP-12*$\mathcal{K}^{\nabla}$	1	12,335	12.3	0.5
EP-13	14	24,621	10.8	1.2	*EP-13*$\mathcal{K}^{\nabla}$	1	24,626	14.9	1.1
EP-14	15	49,200	14.5	2.9	*EP-14*$\mathcal{K}^{\nabla}$	1	49,205	20.9	2.5
EP-15	16	98,355	22.7	6.9	*EP-15*$\mathcal{K}^{\nabla}$	1	98,360	34.7	5.5
EP-16	17	196,662	39.5	16.3	*EP-16*$\mathcal{K}^{\nabla}$	1	196,667	59.2	12.2
EP-17	18	393,273	72.5	38.4	*EP-17*$\mathcal{K}^{\nabla}$	1	393,278	109.5	27.8
EP-18	19	786,492	140.2	89.8	*EP-18*$\mathcal{K}^{\nabla}$	1	786,497	211.7	61.7
EP-19	20	1,572,927	273.6	210.7	*EP-19*$\mathcal{K}^{\nabla}$	1	1,572,932	419.8	135.8
EP-20	21	3,145,794	556.3	485.8	*EP-20*$\mathcal{K}^{\nabla}$	1	3,145,799	826.5	310.1
EP-21	22	6,291,525	1,105.3	1,123.7	*EP-21*$\mathcal{K}^{\nabla}$	1	6,291,530	1,672.6	658.4
EP-22	23	12,582,984	2,235.4	2,578.2	*EP-22*$\mathcal{K}^{\nabla}$	1	12,582,989	3,332.5	1,440.0
EP-23	24	25,165,899	4,462.5	5,921.0	*EP-23*$\mathcal{K}^{\nabla}$	1	25,165,904	6,783.9	3,078.9
EP-24	24	–	–	–	*EP-24*$\mathcal{K}^{\nabla}$	1	50,331,731	12,506.5	6,640.1

TABLE B.4: Detailed results for the case study EP.

Network	C	kStates	M	$t(s)$	Network	C	kStates	M	$t(s)$
FS-2	8	3,058.91	315.1	98.0	*FS-2*$\mathcal{K}^{\nabla}$	5	3,505.05	5,656.7	95.0
FS-3	9	3,194.52	327.9	120.7	*FS-3*$\mathcal{K}^{\nabla}$	5	3,533.14	5,671.4	100.2
FS-4	10	3,437.60	360.0	160.4	*FS-4*$\mathcal{K}^{\nabla}$	5	3,561.18	5,686.0	106.2
FS-5	11	3,895.72	404.9	231.2	*FS-5*$\mathcal{K}^{\nabla}$	5	3,589.22	5,700.3	111.4
FS-6	12	4,783.92	501.4	386.8	*FS-6*$\mathcal{K}^{\nabla}$	5	3,617.26	5,715.3	117.5
FS-7	13	6,532.28	669.4	704.8	*FS-7*$\mathcal{K}^{\nabla}$	5	3,645.30	5,728.5	121.8
FS-8	14	10,000.96	1,041.5	1,503.3	*FS-8*$\mathcal{K}^{\nabla}$	5	3,673.34	5,742.6	127.5
FS-9	15	16,910.28	1,844.4	3,300.7	*FS-9*$\mathcal{K}^{\nabla}$	5	3,701.38	5,756.8	134.2
FS-10	16	–	–	–	*FS-10*$\mathcal{K}^{\nabla}$	5	3,729.42	5,772.2	140.6
					FS-15$\mathcal{K}^{\nabla}$	5	3,869.62	5,845.4	173.9
					FS-20$\mathcal{K}^{\nabla}$	5	4,009.82	5,920.3	210.8
					FS-25$\mathcal{K}^{\nabla}$	5	4,150.02	5,994.9	249.3
					FS-30$\mathcal{K}^{\nabla}$	5	4,290.22	6,069.6	297.6
					FS-35$\mathcal{K}^{\nabla}$	5	4,430.42	6,144.5	346.6
					FS-40$\mathcal{K}^{\nabla}$	5	4,570.62	6,240.8	395.3
					FS-45$\mathcal{K}^{\nabla}$	5	4,710.82	6,313.3	461.0
					FS-50$\mathcal{K}^{\nabla}$	5	4,851.02	6,390.4	525.6
					FS-55$\mathcal{K}^{\nabla}$	5	4,991.22	6,464.9	586.3
					FS-60$\mathcal{K}^{\nabla}$	5	5,131.42	6,542.1	659.4
					FS-65$\mathcal{K}^{\nabla}$	5	5,271.62	6,620.3	726.8
					FS-70$\mathcal{K}^{\nabla}$	5	5,411.82	6,699.0	802.5
					FS-75$\mathcal{K}^{\nabla}$	5	5,552.02	6,777.5	880.3
					FS-80$\mathcal{K}^{\nabla}$	5	5,692.22	6,855.5	967.2
					FS-85$\mathcal{K}^{\nabla}$	5	5,832.42	6,933.7	1,062.4
					FS-90$\mathcal{K}^{\nabla}$	5	5,972.62	7,013.6	1,166.5
					FS-95$\mathcal{K}^{\nabla}$	5	6,112.82	7,090.5	1,274.1
					FS-100$\mathcal{K}^{\nabla}$	5	6,253.02	7,170.7	1,368.7
					FS-105$\mathcal{K}^{\nabla}$	5	6,393.22	7,250.1	1,494.3
					FS-110$\mathcal{K}^{\nabla}$	5	6,533.42	7,330.1	1,630.4
					FS-115$\mathcal{K}^{\nabla}$	5	6,673.62	7,411.7	1,748.1
					FS-120$\mathcal{K}^{\nabla}$	5	6,813.82	7,490.8	1,887.5
					FS-125$\mathcal{K}^{\nabla}$	5	6,954.02	7,562.4	2,075.7
					FS-126$\mathcal{K}^{\nabla}$	5	6,982.76	7,580.9	2,102.4

TABLE B.5: Detailed results for the case study FS.

Network	C	kStates	M	$t(s)$	Network	C	kStates	M	$t(s)$
PG-1	5	0.26	5.8	0.0	*PG-1*$\mathcal{K}^{\nabla}$	3	0.51	6.1	0.0
PG-2	6	0.62	5.8	0.0	*PG-2*$\mathcal{K}^{\nabla}$	3	1.07	6.2	0.0
PG-3	7	1.42	5.9	0.0	*PG-3*$\mathcal{K}^{\nabla}$	3	2.31	6.4	0.0
PG-4	8	3.16	6.0	0.1	*PG-4*$\mathcal{K}^{\nabla}$	3	5.05	6.6	0.0
PG-5	9	6.98	6.2	0.7	*PG-5*$\mathcal{K}^{\nabla}$	3	11.05	6.9	0.1
PG-6	10	15.23	6.6	4.1	*PG-6*$\mathcal{K}^{\nabla}$	3	24.11	7.5	0.4
PG-7	11	33.03	7.3	22.9	*PG-7*$\mathcal{K}^{\nabla}$	3	52.34	8.6	1.0
PG-8	12	71.18	8.7	143.5	*PG-8*$\mathcal{K}^{\nabla}$	3	99.18	10.7	2.1
PG-9	13	152.59	11.7	704.5	*PG-9*$\mathcal{K}^{\nabla}$	3	242.81	15.6	5.9
PG-10	14	325.65	17.7	3,688.6	*PG-10*$\mathcal{K}^{\nabla}$	3	455.79	24.1	12.1
PG-11	15	692.24	29.5	17,763.6	*PG-11*$\mathcal{K}^{\nabla}$	3	1,106.05	45.1	31.8
PG-12	16	–	–	–	*PG-12*$\mathcal{K}^{\nabla}$	3	2,060.41	82.4	64.6
					PG-13$\mathcal{K}^{\nabla}$	3	4,964.49	174.9	166.8
					PG-14$\mathcal{K}^{\nabla}$	3	9,191.55	335.9	333.0
					PG-15$\mathcal{K}^{\nabla}$	3	22,020.24	722.0	854.6
					PG-16$\mathcal{K}^{\nabla}$	3	46,203.03	1,479.1	1,910.6
					PG-17$\mathcal{K}^{\nabla}$	3	96,731.29	3,034.9	4,165.8
					PG-18$\mathcal{K}^{\nabla}$	3	202,113.18	6,234.1	9,179.0

TABLE B.6: Detailed results for the case study PG.

Network	C	kStates	M	$t(s)$	Network	C	kStates	M	$t(s)$
TT-1	2	0.03	5.9	0.0	*TT-1*$\mathcal{K}^{\nabla}$	1	0.03	6.3	0.0
TT-2	3	1.04	6.1	0.0	*TT-2*$\mathcal{K}^{\nabla}$	1	1.04	6.9	0.0
TT-3	4	6.88	6.6	0.0	*TT-3*$\mathcal{K}^{\nabla}$	1	7.02	7.8	0.0
TT-4	5	49.18	9.3	0.8	*TT-4*$\mathcal{K}^{\nabla}$	1	50.40	12.7	0.6
TT-5	6	319.23	25.8	6.7	*TT-5*$\mathcal{K}^{\nabla}$	1	327.17	42.5	5.2
TT-6	7	1,873.07	123.1	36.3	*TT-6*$\mathcal{K}^{\nabla}$	1	1,916.68	217.0	26.9
TT-7	8	10,847.61	625.3	236.3	*TT-7*$\mathcal{K}^{\nabla}$	1	11,054.97	1,232.3	197.2

TABLE B.7: Detailed results for the case study TT.

Bibliography

[1] H. Kopetz, A. Ademaj, et al. "*The Time-Triggered Ethernet (TTE) Design*". In: *ISORC*. IEEE, 2005, pp. 22–33.

[2] T.S. Rappaport. *Wireless Communications*. Vol. 2. Prentice Hall, 2002, pp. 453,454.

[3] E.-R. Olderog and H. Dierks. *Real-Time Systems – Formal Specification and Automatic Verification*. Cambridge University Press, 2008, pp. 1–320.

[4] C. Herrera, B. Westphal, et al. "*Reducing Quasi-Equal Clocks in Networks of Timed Automata*". In: *FORMATS*. Vol. 7595. LNCS. Springer, 2012, pp. 155–170.

[5] G. Behrmann, A. David, et al. "*A Tutorial on Uppaal*". In: *SFM*. Vol. 31-85. LNCS. Springer, 2004, pp. 200–236.

[6] C. Baier and JP. Katoen. *Principles of Model Checking*. The MIT Press, 2008, pp. 1–975.

[7] C. Herrera, B. Westphal, et al. "*Quasi-Equal Clock Reduction: More Networks, More Queries*". In: *TACAS*. Vol. 8413. LNCS. Springer, 2014, pp. 295–309.

[8] C. Herrera and B. Westphal. "*Quasi-Equal Clock Reduction: Eliminating Assumptions on Networks*". In: *HVC*. Vol. 9434. LNCS. Springer, 2015, pp. 173–189.

[9] C. Herrera and B. Westphal. "*The Model Checking Problem in Networks with Quasi-Equal Clocks*". In: *TIME*. IEEE, 2016, pp. 21–30.

[10] S. Feo-Arenis, A. Podelski, et al. "*Ready for Testing: Ensuring Conformance to Industrial Standards Through Formal Verification*". In: *Formal Asp. Comput.* 28.3 (2016), pp. 499–527.

[11] J. Bengtsson and W. Yi. "*Timed Automata: Semantics, Algorithms and Tools*". In: *ACPN*. Vol. 3098. LNCS. Springer, 2003, pp. 87–124.

[12] M. Muñiz, B. Westphal, and A. Podelski. "*Detecting Quasi-Equal Clocks in Timed Automata*". In: *FORMATS*. Vol. 8053. LNCS. Springer, 2013, pp. 198–212.

[13] D. Dietsch, A. Podelski, et al. "*Disambiguation of Industrial Standards Through Formalization and Graphical Languages*". In: *RE*. IEEE, 2011, pp. 265–270.

[14] S. Gobriel, S. Khattab, et al. "*RideSharing: Fault Tolerant Aggregation in Sensor Networks Using Corrective Actions*". In: *SECON*. IEEE, 2006, pp. 595–604.

[15] H. E. Jensen, K. G. Larsen, et al. "*Modelling and Analysis of a Collision Avoidance Protocol using SPIN and Uppaal*". In: *DIMACS*. Vol. 32. DIMACS. 1996, pp. 33–50.

[16] P. Kordy, R. Langerak, et al. "*Re-verification of a Lip Synchronization Protocol Using Robust Reachability*". In: *FMA*. Vol. 20. EPTCS. 2009, pp. 49–62.

[17] S. Limal, S. Potier, et al. "*Formal Verification of Redundant Media Extension of Ethernet PowerLink*". In: *ETFA*. IEEE, 2007, pp. 1045–1052.

[18] W. Steiner and W. Elmenreich. "*Automatic Recovery of the TTP/A Sensor/Actuator Network*". In: *WISES*. Vienna University of Technology, 2003, pp. 25–37.

[19] K. Godary. "*Validation temporelle de réseaux embarqués critiques et fiables pour l'automobile*". PhD thesis. Institut National des Sciences Appliquées de Lyon, France, 2004, pp. 20–32.

[20] B. Bérard, P. Bouyer, et al. "*Analysing the PGM Protocol with UPPAAL*". In: *IJPR* 42.14 (2004), pp. 2773–2791.

[21] N. Petalidis. "*Verification of a Fieldbus Scheduling Protocol Using Timed Automata*". In: *CI* 28.5 (2009), pp. 655–672.

[22] C. Daws and S. Yovine. "*Reducing the Number of Clock Variables of Timed Automata*". In: *RTSS*. IEEE, 1996, pp. 73–81.

[23] C. Daws and S. Tripakis. "*Model Checking of Real-Time Reachability Properties Using Abstractions*". In: *TACAS*. Vol. 1384. LNCS. Springer, 1998, pp. 313–329.

[24] É. André. "*Dynamic Clock Elimination in Parametric Timed Automata*". In: *FSFMA*. OASICS. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 18–31.

[25] G. Behrmann, P. Bouyer, et al. "*Static Guard Analysis in Timed Automata Verification*". In: *TACAS*. Vol. 2619. LNCS. Springer, 2003, pp. 254–277.

[26] S. Guha et al. "*Reducing Clocks in Timed Automata While Preserving Bisimulation*". In: *CONCUR*. Vol. 8704. LNCS. Springer, 2014, pp. 527–543.

[27] R. Salah, M. Bozga, et al. "*Compositional Timing Analysis*". In: *EMSOFT*. ACM, 2009, pp. 39–48.

[28] L. Waszniowski and Z. Hanzalek. "*Over-Approximate Model of Multitasking Application Based on Timed Automata Using Only One Clock*". In: *IPDPS*. IEEE, 2005, pp. 128a–128a.

[29] A. Janowska and W. Penczek. "*Path Compression in Timed Automata*". In: *Fundam. Inform.* 79.3-4 (2007), pp. 379–399.

[30] V. Braberman, D. Garbervetsky, et al. "*Speeding Up Model Checking of Timed-Models by Combining Scenario Specialization and Live Component Analysis*". In: *FORMATS*. Vol. 5813. LNCS. Springer, 2009, pp. 58–72.

[31] V. Braberman et al. "*Improving the Verification of Timed Systems Using Influence Information*". In: *TACAS*. Vol. 2280. LNCS. Springer, 2002, pp. 21–36.

[32] F. He, H. Zhu, et al. "*Compositional Abstraction Refinement for Timed Systems*". In: *TASE*. IEEE, 2010, pp. 168–176.

[33] H. Dierks, S. Kupferschmid, et al. "*Automatic Abstraction Refinement for Timed Automata*". In: *FORMATS*. Vol. 4763. LNCS. Springer, 2007, pp. 114–129.

[34] K. Okano, B. Bordbar, et al. "*Clock Number Reduction Abstraction on CEGAR Loop Approach to Timed Automaton*". In: *ICNC*. IEEE, 2011, pp. 235–241.

[35] D. Lugiez, P. Niebert, et al. "*A Partial Order Semantics Approach to the Clock Explosion Problem of Timed Automata*". In: *TACAS*. Vol. 2988. LNCS. Springer, 2004, pp. 296–311.

[36] J. Zhao, H. Xu, et al. "*Partial Order Path Technique for Checking Parallel Timed Automata*". In: *FTRTFT*. Vol. 2469. LNCS. Springer, 2002, pp. 417–432.

[37] E.-R. Olderog and M. Swaminathan. "*Layered Composition for Timed Automata*". In: *FORMATS*. Vol. 6246. LNCS. Springer, 2010, pp. 228–242.

[38] J. Bengtsson, B. Jonsson, et al. "*Partial Order Reductions for Timed Systems*". In: *CONCUR*. Vol. 1466. LNCS. Springer, 1998, pp. 485–500.

[39] M. Minea. "*Partial Order Reduction for Model Checking of Timed Automata*". In: *CONCUR*. Vol. 1664. LNCS. Springer, 1999, pp. 431–446.

[40] J. Håkansson and P. Pettersson. "*Partial Order Reduction for Verification of Real-Time Components*". In: *FORMATS*. Vol. 4763. LNCS. Springer, 2007, pp. 211–226.

[41] R. Alur and D. Dill. "*A Theory of Timed Automata*". In: *TCS* 126.2 (1994), pp. 183–235.

[42] M. Muñiz, B. Westphal, and A. Podelski. "*Timed Automata with Disjoint Activity*". In: *FORMATS*. Vol. 7595. LNCS. Springer, 2012, pp. 188–203.

[43] E.-R. Olderog and M. Swaminathan. "*Structural Transformations for Data-Enriched Real-Time Systems*". In: *IFM*. Vol. 7940. LNCS. Springer, 2013, pp. 378–393.

[44] W. Janssen, M. Poel, et al. "*Layering of Real-Time Distributed Processes*". In: *FTRTFT*. Vol. 863. LNCS. Springer, 1994, pp. 393–417.

[45] P. Bouyer. "*Forward Analysis of Updatable Timed Automata*". In: *Formal Methods in System Design* 24.3 (2004), pp. 281–320.

[46] C. Herrera, B. Westphal, A. Podelski, et al. "*Quasi-Dependent Variables in Hybrid Automata*". In: *HSCC*. ACM, 2014, pp. 93–102.

[47] T. A. Henzinger, X. Nicollin, et al. "*Symbolic Model Checking for Real-Time Systems*". In: *Inf. Comput.* 111.2 (1994), pp. 193–244.

[48] R. Alur, L. Fix, et al. "*Event-Clock Automata: A Determinizable Class of Timed Automata*". In: *Theor. Comput. Sci.* 211.1-2 (1999), pp. 253–273.

[49] G. Geeraerts, J.-F. Raskin, et al. "*On Regions and Zones for Event-Clock Automata*". In: *Formal Methods in System Design* 45.3 (2014), pp. 330–380.

[50] P. Bouyer, C. Dufourd, et al. "*Updatable Timed Automata*". In: *Theor. Comput. Sci.* 321.2-3 (2004), pp. 291–345.

[51] P. Bouyer and F. Chevalier. "*On Conciseness of Extensions of Timed Automata*". In: *Journal of Automata, Languages and Combinatorics* 10.4 (2005), pp. 393–405.

[52] G. Norman, D. Parker, et al. "*Model Checking for Probabilistic Timed Automata*". In: *Formal Methods in System Design* 43.2 (2013), pp. 164–190.

[53] G. Morbé, F. Pigorsch, and C. Scholl. "*Fully Symbolic Model Checking for Timed Automata*". In: *CAV*. Vol. 6806. LNCS. Springer, 2011, pp. 616–632.

[54] S. Bogomolov, C. Herrera, et al. "*Verification of Fault-Tolerant Clock Synchronization Algorithms. Benchmark Proposal.*" In: *ARCH@CPSWeek*. Vol. 43. EPiC Series in Computing. EasyChair, 2016, pp. 36–41.