Compositional verification of component-based real-time systems and applications

Souha Ben Rayana - Tekaya

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Souha Ben Rayana - Tekaya

Thèse dirigée par **Saddek Bensalem**
et codirigée par **Marius Bozga**

préparée au sein **du Laboratoire Verimag**
et de **l'Ecole Doctorale Mathématique, Sciences et Technologies de l'Information, Informatique (MSTII)**

Compositional Verification of Component-Based Real-Time Systems and Applications

Thèse soutenue publiquement le **4 Novembre 2016**,
devant le jury composé de :

Mme, Florence Maraninchi
Professeur, Université Grenoble Alpes et INP Grenoble , Présidente
M, Ahmed Bouajjani
Professeur, Université Paris Diderot, Rapporteur
M, Kim Guldstrand Larsen
Professeur, Université D'Aalborg , Rapporteur
Mme, Erika Ábrahám
Professeur, Université de RWTH Aachen , Examinateur
M, Doron Angel Peled
Professeur, Université de Bar Ilan, Examinateur
M, Natarajan Shankar
Docteur, SRI International, Examinateur
M, Saddek Bensalem
Professeur, Université Grenoble Alpes, Directeur de thèse
M, Marius Bozga
Ingénieur de recherche - HDR, CNRS, Co-Directeur de thèse

Acknowledgments

Foremost, I would like to express my special thanks to my supervisor Professor Saddek Bensalem. I would like to thank him for offering me ideal research conditions and for highlighting valuable work directions. I am deeply grateful to my supervisor, Dr. Marius Bozga for his encouragement and for his precious suggestions and advices. Without his involvement, this work would not have been the same.

I would also like to thank my PhD committee members notably the reviewers, Professor Ahmed Bouajjani and Professor Kim Guldstrand Larsen, and Professor Erika Ábrahám for their valuable suggestions to improve this manuscript. I thank my PhD committee members for making my defense an enjoyable moment, for their brilliant comments and for highlighting important research perspectives.

I would like to express my special appreciation and thanks to my co-authors Lăcrămioara Aştefănoaei and Jacques Combaz for the fruitful collaboration.
I would like to thank all my friends at Verimag for the exellent work environment, as well as the administrative staff for their help.

I thank my teacher in the first-year of primary school for telling my mother, after my low results especially in mathematics, that I do not have abstraction skills. Thanks to her, my mother provided me with high care and made me miraculously a different student in a matter of months.

I am grateful to my mother and my father to whom I owe my success. I thank them for their encouragment, sacrifices and for teaching us the importance pf knowledge.

I thank my family: my sister Wafa, my brothers and my sisters-in-law for their continous encouragement.

I am grateful to my family-in-law, especially to my mother-in-law for taking care of us: me and my little baby boy. Finally, I thank my husband *Slim* for his patience and for being unconditionally supportive and caring.

To my family,
And to the memory of those who passed away during my doctoral studies:
My kindhearted aunt Faiza,
My loving aunt Habiba,
And my cousin, Dr. Yassine, who left us very early without a goodbye.

Compositional Verification of Component-based Real-time Systems and Applications

Abstract:
Compositional Verification aims at breaking down the complexity of the verification task by relying on the separate analysis of the sub-components and inferring global properties of the system from their local properties. In the framework of real-time systems, one main obstacle for developing fully compositional methods is the synchronous model of time.
We propose a verification method based on the deductive approach where the set of the reachable states of the system is over-approximated by an invariant computed in a fully compositional manner. It comprises local component invariants and an interaction invariant characterizing the interactions between the components. In addition, we introduce auxiliary clocks, called *history clocks*, which allow to automatically generate new invariants capturing the constraints induced by the time-synchronizations between the different components. We completed this compositional invariant generation approach with a counterexample-based invariant enforcement module analyzing iteratively the generated counterexamples.
Besides its scalability, the method can be extended to the uniform verification of parameterized timed systems.

Our compositional verification method was implemented in the RTD-Finder tool. The experimental results show that the verification time for large systems is drastically reduced in comparison with exploration techniques, especially when the global invariant catches the safety property of interest.
Keywords: verification, timed automata, real-time, component-based, compositional, deductive, invariant, safety property

Vérification Compositionnelle des Systèmes Temps-Réels à Base de Composants et Applications

Résumé:
On s'intéresse à la vérification formelle des propriétés de sûreté pour les systèmes temps-réels à base de composants. Le but est de proposer une alternative aux techniques d'exploration où le produit de tous les composants d'un système donné est calculé, résultant en une complexité exponentielle de vérification rendant souvent impossible la vérification de larges systèmes.

La vérification compositionnelle a pour but d'alléger la complexité de vérification du système en comptant sur l'analyse locale de ses composants. Les propriétés globales en sont ensuite déduites. Dans le cas des systèmes temps-réels, une difficulté majeure pour le développement d'une approche compositionnelle consiste au modèle synchrone du temps où les horloges des différents composants avancent simultanément. En effet, cet aspect est difficile à considérer dans un cadre compositionnel. Nous proposons une méthode basée sur l'approche déductive et consistant à calculer d'une manière purement compositionnelle une sur-approximation de l'ensemble des états atteignables du système à travers un invariant. Ce dernier se compose d'invariants locaux propres aux composants et un invariant d'interaction caractérisant les interactions entre eux. En plus, pour considérer le modèle synchrone du temps, nous introduisons des horloges auxiliaires appelées " Horloges d'Histoire ". Elles permettent de générer des invariants supplémentaires permettant de détecter des relations induites par les synchronisations temporelles des différents composants. Appliqué à plusieurs exemples de systèmes, l'invariant s'est avéré souvent suffisamment fort avec une réduction importante de la complexité de vérification. Toutefois, puisque la méthode est basée sur une sur-approximation, des faux contre-exemples peuvent être générés. Nous avons complété la méthode avec un module destiné pour leur analyse.

Au delà de son passage à l'échelle, la méthode est étendue pour la vérification uniforme des systèmes paramétrés, où certains composants sont identiques. La validité de la propriété peut être affirmée indépendamment de leur nombre.
Cette méthode est implémentée dans l'outil RTD-Finder conçu pour la vérification des systèmes modélisés au langage BIP (Behavior-Interaction-Priority). Les résultats d'expérimentation montrent la réduction de la complexité de vérification en

comparaison avec l'approche monolithique, surtout quand l'invariant global est en mesure de détecter la propriété d'intérêt.

Mots-clés: vérification, automate, temps-réels, composant, modèle, déductive, invariant, sûreté

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the last decades, the usage of information and communication systems has been spectacularly increasing. These systems become more and more complex and their omnipresence increases as a result of the embedded systems conquering our daily lives. We cite as examples the smart-phones, the medical devices and the transportation and multimedia systems. Even though technological advances try to enhance continuously the quality of service of such systems and reduce for instance the response time, there are high priority and urge for establishing error absence especially for systems with high degree of criticality. The presence of unexpected errors can have dramatic impacts, ranging from significant financial consequences for the producer to the loss of human lives. We mention for example the tragic consequences due to a conversion of a 64-bit floating point into a 16-bit integer value in the Ariane 5 maiden flight [Rep96].
Formal methods attempt to offer theories, methods and tools to ensure the correct design of systems. The substantial task of validating the correctness is confronted with the increasing complexity of the systems combined with the pressure to reduce the system construction time. Therefore, the computational complexity induced by the state space is a major hurdle in constructing large complex systems. Furthermore, in order to accurately express real life situations and to specify safety time bounds for the occurrence of events, some systems- the so-called *real-time* systems- evolve to handle timing constraints. In *time-critical* systems, the correctness does not rely only on the insurance of a desired logical output for a given input but also on the time on which the result is generated. Real-time

systems manifest another limiting character for correctness checking. It consists in the presence of additional concurrent temporal observations expressed by the local clocks of the different components. This induces a larger and more complex reachable state space increasing the complexity issue and revealing the need for adequate theories and methods for modeling, specification and correctness checking.
Generally, for timed or untimed systems, there are two main approaches for detecting a requirement violation:

- **Formal testing** is a dynamic method consisting in providing different input values and feeding them to the system for execution in order to confirm that the results respect the desired requirements. Only a complete testing can assert the absence of property violation, meaning that the testing scenarios should be exhaustively covered. Unfortunately, in practice, the number of all possible execution paths is hard, if not impossible, to test. On the other side, the testing of a partial set of execution paths can at best attest of the presence of requirement violation but cannot give evidence of its absence. Dijkstra gave a postulation on this in 1969.

  > *“ Program testing can be used to show the presence of bugs, but never to show their absence. ”*
  >
  > Edsger Dijkstra , *[BR70]* , 1969

- **Formal verification** Regarding the coverage limitation of the testing methodology, more complete alternatives have been proposed. The most investigated approach, formal verification, relies upon perceiving systems as mathematical objects with well-defined behaviors. It allows to reduce the requirement checking to a mathematical problem aiming to verify if the behavior of a given system satisfies a specification formulated in mathematical logic. Formal verification allows to attest that the system is correct by use of mathematically rigorous theories and tools, else provides the input scenarios that violate the requirements if exist. Another advantage of formal verification is that it can be performed at relatively early stages as it is applied to a system model. At such stages, the costs of the errors' correction is less important.
  In the past three decades, promising formal verification techniques and tools have been developed. At the beginning, in the early 80's, the most common framework for verification was theorem-proving where the system and requirement are expressed in a formal system defined by axioms and

inference rules. Theorem proving consists in inducing the property of the system from its axioms by use of the rules. While many theorem prover tools have been developed, the lack of automation is the main shortcoming of the proof-theoretic reasoning since the search for the proofs may require the interaction with a human.

Static analysis has shown big successes in the field of software verification. It relies on a family of techniques for automatically examining a program and getting information about its behavior without executing it. The behaviors of the program can be abstracted into a decidable overapproximation or underapproximation, then an attempt is conducted to prove the property in the abstract version of the program. Cousot and Cousot [CC77] introduced abstract interpretation as a framework to relating abstract analyses to the program execution. It consists in defining a suitable abstract domain of computation of the program and then interpreting a program relative to this domain. A discrete approximation applied to the semantics of the C programming language was implemented in the Astree tool [BCC+02]. For instance, it proved the absence of errors for 132000 lines of flight control software in less than an hour. As other commercial successes of static analysis, we mention SLAM [BMMR01] and Java PathFinder [VHB+03].

In general, static analysis methods traded precision for efficiency by using abstraction and by merging abstract states at join points.

This limitation led to a significant research focusing on another instance of formal verification called *model-checking*. It offers a fully automatic method for verifying finite-state systems through an efficient graph search in order to decide whether or not the requirement holds over the system's state graph. The model-checking approach is algorithmic and is able to provide a counterexample in case of property violation. Nevertheless, the model-checking is hampered by the combinatorial explosion problem since an exhaustive search should be performed over the state-space of the system's model. The complexity prevails even more in case of timed systems possessing additional temporal observations.

Despite having been of prominent outcome in industrial hardware and software community, model-checking has other shortcomings besides the state-space explosion problem.

- Regarding the exploration of the entire state-space, explicit-state model checking is not suitable for infinite-state systems. There is a need for abstraction techniques to make the space graph to explore finite.

- It does not allow the verification of generalizations, for instance the parameterized timed systems containing an arbitrary number of similar components.
- In case a counterexample is detected, it may be required to reconstruct the global system.

In an introductory paper to formal verification, published 20 years ago, the future directions for improvements were stated by Clarke as follows.

> *“ Significant advances in the practical use of formal methods have relied on fundamental results drawn from all areas in computer science, not necessarily directly intended for formal methods. Further work needs to be done in the following areas: composition, decomposition, abstraction, reusable models and theories, combination of mathematical theories, and data structures and algorithms. ”*
>
> Edmund Clarke , *[CW96]* , 1996

In the following, we briefly relate the state-of-the-art of the advances on formal verification and introduce in particular model-checking improvements. Besides, we mention the existing proposals for handling real-time systems.

1.2 Improvements in Model-Checking

1.2.1 Reduction Techniques

In order to circumvent the state-space explosion problem inherent to model checking, reduction techniques are investigated to construct a smaller state space to search by algorithms. The theoretical foundations for such reductions rely mainly on equivalences or abstraction. In fact, a considerable family of reduction techniques relies on bisimulation equivalence [Mil89] which is a binary relation denoting the possibility for two state transition systems to simulate each other, that is the possibility for every step in the one to be matched to one or more steps from the other. If two state spaces are bisimilarly equivalent, then checking the more complex one can be reduced to model-checking the other.

1.2.1.1 Partial Order Reduction

An execution of actions is said to be totally ordered if every action is compelled to follow and precede precise actions. At the opposite, the partial ordering property allows that some actions occur in different orders. Actions of an execution fragment are independent if their swapping leads up to an execution fragment starting and ending with the same states. Several strategies have been conducted in order to exploit the commutativity of independant actions in partially ordered fragments. We cite the Vlmari's stubborn sets [Val89] and Godefroid's persistent sets [God90, God96] methods. The Ample sets method of Peled [Pel93, HP94, Pel94] focuses on the modification of the state space generation algorithm. Usually, while building the state graph of a given transition system, the successors of a state are computed by consideration of all the possible actions. The Ample set method proposes to explore instead a subset of the action set in the hope of buidling a transition system that is drastically smaller and with a relatively small overhead. For the construction of the representative transition system to be sound, it should guarantee that the satisfaction of the property of interest is insensitive to the choice between the original and the reduced systems. Since the partial order reduction technique builds upon the assumption that the components are asynchronous, treating with realistic systems requires the identification of path fragments which exclusively differ on the concurrently executed actions.

The major obstacle encountered in the case of real-time systems is that the clocks of the different components are implicitly synchronized since they do all advance at the same rate. Therefore, even in the absence of explicit synchronization between the components, different interleavings between independent actions would engender different combinations of clocks valuations. In [BJLY98] [Min99], a solution was proposed at the aim of applying standard partial order reduction techniques in spite of the implicit synchronization of clocks. The basic idea was to *desynchronize* the clocks, that is to let them run without synchronization, until communication time. This solution was presented by a local-time semantics, along with its symbolic version. The PORT extension [HCM+08] of the UPPAAL tool implements these partial order reduction techniques to real-time systems.

1.2.1.2 Symmetry Reduction

In a concurrent system with many replicated components, the inherent symmetry is reflected in the state space. It can result in subspaces of the full state set being equivalent up to the rearrangement of components. Thereby, symmetry can serve

as a basis to exploit the automorphisms of the system's global state-space and hence reduce its exploration during verification. Several strategies have been conducted at this aim and some have been successfully implemented in model-checkers such as SMV [McM93], Murφ [ID96], and SPIN [Hol97, BDH01]. Symmetry reduction was also extended for timed systems. In [Hen02], an extension to timed systems of the method of [ID96] is proposed and is proven to be sound. It begins with the introduction of the new data type, scalarsets, allowing the user to explicitly stress the symmetry in the model. This is followed by the extraction of automorphisms on the state graph of the system model. This method was successfully implemented in the UPPAAL tool [HBL+03] and offered a drastic reduction of both computation time and memory usage. Also, in the framework of real-time systems, Wang et al. exploit symmetry by introduction of a BDD-like [Bry86] data structure implemented in the RED tool [Wan00, WS02], but there an over-approximation of the reachable states set is computed.

1.2.2 Approximation

At the aim of avoiding the full state-space exploration, several works based on underapproximations and overapproximations were conducted.
In the hope of exploiting the rich progress in the verification of untimed systems, Tripakis and Yovine [TY01] used a technique based on bisimulation to approximate timed systems with untimed automata.
Abstraction is another strategy counting on an over-approximation of the reachable state set and has been widely studied to timed systems and hybrid systems in general. It combines the model-checking with finite-state abstraction. The advantage of abstraction is to map the original model into a less complex model conserving the behavior of concern, by merging 'similar' states. The *CEGAR* (Counterexample-Guided Abstraction Refinement) was first proposed for finite-state systems and offers a method for the refinement and validation of the generated counterexamples [CGJ+00]. It starts from an initial abstraction together with the property to check and begins by searching for the parts of the model that violate it. Afterwards, it validates and refines iteratively the counterexamples until one of them is proven to be valid or until no further counterexamples are generated, in which case the validity of the property is asserted. This approach differs from the usual abstraction techniques in that the property of interest is considered while building the initial abstraction. The CEGAR approach was extended to infinite systems and to hybrid systems in particular [CFH+03, FCJK05]. In [JM06], Craig interpolation and lazy abstraction serve as basis for model-checking infinite-state sequential programs.

In the field of timed automata, we cite the work in [KP07] based on SAT-solving and an extension of CEGAR with syntactic information about Craig interpolants. In [LL98], the proposed method starts with a small subset of the automata in order to build an over-approximation and refines it by adding iteratively other automata. The works in [DKL07], [PDMV13] differ from previous CEGAR approaches for timed and hybrid systems in that the abstractions have also the form of hybrid automata.

Clock-related Abstractions

Several research works have been conducted to circumvent the complexity issues specific to timed systems and induced by the continous timed observations that are the clocks. In [DY96], two algorithms were proposed at the aim of reducing the number of clocks of a timed automaton. One of them consists in detecting sets of clocks that are always equal and the other proposes to detect the clocks whose values are not relevant for system evolution.
By essence, timed automata have infinite state space, whereas algorithmic verification requires exact finite abstractions. Most of verification tools for timed systems (*eg.* KRONOS [BDM+98] and UPPAAL[LPY97] use abstractions based on zones at the aim of ensuring finiteness. One classical abstraction method is extrapolation and it considers the maximum constants to which clocks are compared. It merges states which are identical except from the clock values which exceed these maximum constants. In [BBFL03], the authors suggest to make the maximum constants depend not only of the particular clock but also of the particular location of the timed automaton. In [BBLP06], maximal lower and upper bounds are distinguished and coarser abstractions are obtained. Besides, the latter abstraction is complete for both reachability and liveness properties.
In [DT98], other abstraction methods were proposed. The inclusion abstraction maps subsets of concrete states to a same abstract state, allowing to reduce the total number of symbolic states.
In order to overcome the infinite-state problem inherent to timed systems, Wong-Toi [WT95] proposed to collapses symbolic states sharing the same location to one symbolic state whose zone is the convex hull of their zones. This method was called *convex-hull approximation* and was implemented in different tools [BDM+98, BDL+06].

1.2.3 Bounded Model Checking

The practical method of bounded model checking (BMC) [BCCZ99] was first proposed by Biere et al. and has gained much interest. It proposes to alleviate the combinatorial explosion problem by settling for the verification of the safety property for a given bound k of execution steps. The technique efficiently reduces to a boolean satisfiability problem and can therefore benefit from the scalability of the modern SAT-solvers which are capable of handling propositional satisfiability problems with great number of variables. BMC was extended to timed systems [ACKS02] where it boils down to deciding the satisfiabilty of a math-formula obtained by combination of propositional variables and linear mathematical relations over the real clock variables. The main disadvantage of BMC is its incompleteness since it searches for the counterexamples refuting the property of interest under the bound k. If no counterexample under this bound violates the property, either k is increased or the algorithm stops for memory shortage, with no certainty about the validity of the property.

1.3 Deductive Approach

The deductive verification method is characterized by that the target system as well as the properties to verify are represented as logical predicates that need to be proven . Their validity is proven by deduction in logical calculus. Deductive verification enables proofs for systems with infinitely many states, in contrast to model checking. To prove that a predicate Ψ is an invariant of the system S, it is sufficient to find an auxiliary predicate Φ^{aux} which satisfies the following conditions:

- Φ^{aux} is valid for the initial states of the system.
- Φ^{aux} is preserved by every transition of the system. That is for every state s of the system satisfying Φ^{aux}, any state s' reachable from s by any transition τ does also satisfy it.
- Φ^{aux} implies Ψ.

If the two first conditions are valid, the predicate Φ^{aux} is called an *inductive invariant* of S. If in addition the third condition is valid, then the predicate Ψ is an invariant of the system, i.e every reachable state s of S satisfies it.

$$Init \Rightarrow \Phi^{aux}$$

$$(R_1)\qquad \frac{\{\Phi^{aux}\}\tau\{\Phi^{aux}\}, \forall \tau \in S \quad \Phi^{aux} \Rightarrow \Psi}{S \models \Box\Psi}$$

In [MP95], it is proven that the rule R_1 is sound for proving invariance properties of transition systems.

However, it does not give a clue on how to compute the auxiliary inductive invariant Φ^{aux}. It keeps open two questions:

1. how to compute the auxiliary Φ^{aux} invariant implying Ψ without reducing to the computation of the reachable state set of the system and
2. how to prove that Φ^{aux} is preserved by each transition of the system and is valid at the initial state.

Our Approach: Overapproximation A recent trend in formal verification is to trade off completeness for automation and complexity avoidance.

The core idea of our methodology lays upon the verification rule (R_1) where an over-approximation of the reachable state set is required. The verification rule in R_1 expresses that if the predicate Φ^{aux} is satisfied by the initial state *Init* and if it is preserved with all the transitions of the system, then every reachable state of the system satisfies it and it is an inductive invariant.

The most intuitive solution for the auxiliary predicate Φ^{aux} is the set of reachable states of the system $Reach(S)$. Yet its computation boils down to the same state-space explosion problem inherent to model-checking. We propose at a first stage to trade off completeness for the avoidance of the complexity issue by relying rather on the computation of an over-approximation $Reach_{app}(S)$ of the set $Reach(S)$. If $Reach_{app}(S)$ implies the property Ψ, then the system satisfies it.

$$\frac{Reach(S) \subseteq Reach_{App}(S), \quad Reach_{app}(S) \Rightarrow \Psi}{S \models \Box\Psi}$$

The main purpose of this work is to find the approximation $Reach_{app}(S)$ in a fully compositional manner.

1.4 Contribution

The main objective of this thesis is the verification of safety properties for real-time systems. It aims at contributing to the applicability of formal verification methods

to realistic systems with large numbers of components. In order to achieve this, we are moving towards the compositional reasoning approach. Compositional verification is aimed to avoid the construction of the product of all the states of the global system. It relies on the decomposition of the verification problem to more manageable sizes. Instead of verifying the system globally, the small components (subsystems) composing the system are first verified, then global properties of the system should be inferred. This is supposed to break down the complexity of the verification task since the size of the subsystems is often quite smaller than the size of the global system thus the state-space explosion is more likely to be avoided. This work lays upon an over-approximation of the reachable state set through the computation of a global invariant to function as $Reach_{app}(S)$ and the application of the above-mentioned deductive verification rule.

Compositional Invariant Generation

Our method aims at inferring global properties of the system from the properties of its components. The extraction of the components' properties is done through local invariant computation, characterizing their behavior in isolation from each other. In our framework, the components don't evolve independently. They are glued together through multi-party interactions synchronizing their actions. This action synchronization restricts the global behavior of the system and constraints the reachable state set. Therefore, in the hope of tightening the over-approximation made by the global invariant, the interaction structure should be considered. This results in additional type of invariant along with the component invariants, focusing on the interaction structure. In [BBSN08, BBNS09], a compositional verification approach was proposed for the verification of component-based untimed systems. It takes advantage of that the invariant of a component is also an invariant of the global system and that the conjunction of invariants is also an invariant. Thereby, $Reach_{app}(S)$ is computed as the conjunction of the following invariants:

- local component invariants: they characterize the behavior of the components in isolation from each other,
- and an interaction invariant: it reflects constraints on the global state set induced by the synchronization of the actions belonging to the different components.

A simple illustration of this appoach is shown in Figure 1.1, where the invariants of components B_1 and B_2 are ϕ_1 and ϕ_2 respectively and the interaction invariant is II_{glue}.

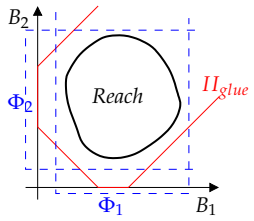


Figure 1.1: Compositional verification of a system of two components

The computed global invariant is rather strong for untimed systems. The method was implemented in a tool, D-Finder [BBNS09], yielding a drastic reduction in the complexity of the verification task.

Compositional Invariant Generation for Timed Systems

Unfortunately, this global invariant is insufficient for timed systems since in no way it takes into account the time synchronization between the components. In fact, in the timed case, the clocks of the different components increase at the same rate, resulting in additional restrictions on the moves of the different components whose transitions are locally guided by timing constraints.
The main contribution of this work is an extension of the above method for timed systems. It offers a proposal to resolve the outstanding issue of capturaing time synchronization of different components in a fully compositional manner. For this purpose, we introduce additional auxiliary clocks, called *history clocks*, allowing to infer the time synchronization induced from the simultaneous execution of actions through interactions. To each action, we introduce an *action history clock* which is reset whenever the corresponding action occurs. Since an interaction synchronizes a set of actions, all of their history clocks are reset simultaneously and hence become equal. On the one hand, this information can be synthesized in an additional type of invariant relating the history clocks. On the other hand, the added history clocks are related to the inner clocks of the components and may be injected in their components' invariants. All in all, the conjunction of the new proposed invariant constraining the history clocks' relations and the local invariants of components extended with history clocks results by transitivity in relations between the original clocks of the different components. In this way, the time synchronization can be inferred in the global invariant, yielding a more successful implementation of the verification rule.

Counterexample-Based Invariant Refinement

The applied verification rule searches for an over-approximation of the reachable state set. If the computed invariant implies the property of interest, then the system satisfies it. Else, a suspected counterexample is generated. It may be the outcome of a behavior in the over-approximation which does not belong to the original model. At a second stage of this work, we propose to analyze the generated counter-examples in order to decide upon their validity. Subsequently, in case the counterexample is spurious, the global invariant is refined with its negation. The iterative analysis of the counterexamples remedies the incompleteness of the method.

Compositional Verification of Parameterized timed systems

Unlike the model-checking which does not handle generalizations, particularly the parameterized timed systems which rely on an arbitrary number of components sharing the same behavior, the compositional invariant generation method can be extended to handle uniform verification of such systems. In the parameterized setting, the proposed invariant can be expressed under a particular shape, allowing the elegant application of a small model theorem. As a result, it suffices to assert the validity of the property of interest for small numbers of components and deduce its validity for all numbers of the replicated component.

Tool Implementation

The proposed compositional verification method was implemented to verify systems modeled in the RT-BIP language [ACS13], where components are modeled by timed automata and are synchronized through multi-party interactions. The developed tool, called RTD-Finder, first computes the global invariant then interacts with a satisfiability checking tool (SAT-solver) to verify if the invariant implies the safety property of interest. This is performed by conjoining the predicate characterizing the violation of the property with the invariant representing the over-approximation of the reachable state set. The tool contains also a counterexample analysis module, aiming at rendering the verification method complete.

1.5 Organization of the Dissertation

The remainder of this thesis is organized as follows:

- Chapter 2 gives an overview of the compositional verification methods proposed in the literature.
- Chapter 3 begins by introducing the BIP modeling framework. Subsequently, the above-mentioned compositional verification method proposed for untimed systems is detailed. Its different modules were implemented in the D-Finder tool which takes as input systems modeled in the untimed BIP language.
- Chapter 4 presents timed component-based systems along with the related syntax and semantics. We conclude this chapter by showing how the D-Finder method, proposed for untimed system, is unable to track time synchronization between components in the timed case. This gives a motivation for the extension of the invariant generation method to handle timed systems.
- Chapter 5 introduces our compositional verification method proposed for timed systems. It mainly builds upon the introduction of the auxiliary history clocks and the extraction of the new invariants intended to offer a more successful application of the verification rule.
- Chapter 6 explains how the generated counterexamples are analyzed and how the global invariant is refined until the validity of the safety property is asserted or a counterexample which truly violates it is revealed.
- Chapter 7 presents the extension of the verification method to parameterized timed systems.
- Chapter 8 shows the implementation details of the RTD-Finder tool.
- Chapter 9 gives the experimental results for different case-studies. They show that the computed invariant is strong enough to detect many safety properties. In such cases, the compositional verification method drastically reduces the verification time for systems with a large number of components.
- We conclude the dissertation in Chapter 10 with an overview of the work and its perspectives.

Chapter 2

Compositional Verification Methods: An overview

> [. . . .] *A captain should endeavor with every art to divide the forces of the enemy, either by making him suspicious of his men in whom he trusted, or by giving him cause that he has to separate his forces, and, because of this, become weaker.*

Machiavelli , *Dell'arte della guerra*, 1520

Contents

Standard monolithic techniques based on the computation of the reachable states set are unpractical since the complexity increases exponentially with the number of components. The compositional reasoning aims to circumvent the state-explosion problem by substituting the single verification over the global state-space by the local analysis of its different components. The verification should allow to infer

global properties of the system from properties of its components. While the local analysis of each component abstracts away the rest of the system, the global properties cannot be inferred without consideration of the interactions gluing them together. In the following of this chapter, we briefly relate some major approaches for compositional verification.

2.1 Assume-guarantee Reasoning

While compositional reasoning aims to analyze the components separately, usually a component satisfies some properties in a specific environment. This led to the research on the assume-guarantee compositional style of reasoning. The works of Owicki and Gries [OG76] and Lamport [Lam77] inspired the introduction of the assume-guarantee compositional methods by Misra and Chandy [MC81], by Jones [Jon83] and by Pnueli [Pnu85]. This approach has been extensively studied in the literature [Sta85, CLM89, CM89, GL91, AL95, AH96, McM97].

Assume-guarantee relies on the separate analysis of components. Suppose that a given system is composed of two components B and B'. The behavior of component B' depends on the behavior of component B. The assume-guarantee method specifies a set of assumptions to be satisfied by B in order to guarantee the correctness of B'. In a mutual manner, establishing the correctness of B requires the validity of a set of assumptions over B'. By appropriately combining a set of assumed and guaranteed properties of B and B', properties of the global system $B||B'$ can be inferred without resorting to the construction of the global state-space. More generally, for a system composed of many components, the local analysis focuses on a single component and abstracts the rest of the system as an *assumption* for this component. Therefore, in the assume-guarantee paradigm, there are two types of properties with respect to a given component: assumptions about the global behavior of its environment and properties that it guarantees if these assumptions are valid. A compositional proof rule should be applied to adequate assumptions which should, on the one hand be obtained by local analysis exclusively and on the one other hand be strong enough to prove the global property of interest.
In the notation of Pnueli, the triple $< f > B < g >$ is interpreted as "whenever B is part of a system satisfying the assumption f, the system must guarantee also the property g".
A basic and typical proof rule relies therefore in the *transitivity* principle and may be expressed by the following inference rule:

$$\text{Rule 2.1}\ \frac{\begin{array}{c} < true > B < A > \\ < A > B' < P > \end{array}}{< true > B||B' < P >}$$

In this inference rule, A denotes an assumption about the environment of component B' whereas P is a property holding for the system resulting from the composition of B and B'.
In the assume guarantee reasoning, a particular attention should be paid to circularity. We consider for illustration the following inference rule:

$$\text{Rule 2.2}\ \frac{\begin{array}{c} < P > B < A > \\ < A > B' < P > \end{array}}{< true > B||B' < P \wedge A >}$$

It is clear that this rule is unsound. The apparent circularity of such premises can be removed by relying on induction over time. Yet, this is applicable to safety properties and not to liveness properties.
Despite being simple, the asymmetric Rule 2.1 is quite useful for checking safety properties, which are the focus of this work. For this rule to be efficient, the assumption A should be smaller than B and it should be at the same time strong enough to ensure that component B' satisfies P. Finding the assumptions manually is unpractical. To remedy this, learning-based techniques may be used.

2.1.1 Automation of Assumption Generation

Learning-Based Techniques In [CGP03], an automatic method to compute the environment assumptions for each component is proposed. It is based on the learning algorithm L^* that was first proposed by Angluin in [Ang87] and ameliorated in [RS93]. It is applied to generate iteratively a finite-state assumption by making queries to a *Teacher* represented by a model checker which answers queries and provides the counterexamples. In this framework, the L^* algorithm targets the computation of the so-called *Weakest Assumption*. By definition, this assumption is characterized by that the premises of the rule are not only sufficient, but should be also necessary for the conclusion of the inference rule to hold. In [GPB05], it has been proven that this assumption exists for any finite-state component and that it can be algorithmically generated. In [AMN05, NMA08], the BDD-based exploration is used to compactly and symbolically implement the assume-guarantee technique through the L^* algorithm. The L^* learning algorithm was first applied to the safety properties verification and the inference Rule 2.1 before being extended to other rules. For instance, in [BGP03], the L^* algorithm targets the circular and symmetric proof rule. There, the number of premises to be

learned increases linearly on the number of components.
The assume-guarantee approach shows a higher efficiency when the interfaces between the components are smaller. Therefore, in [PGB+08], the authors propose to automate the search for a smaller alphabet of the assumption automata that suffices to prove the property. This work resulted in the *alphabet refinement* technique proposed as an extension of the learning based approach in [CGP03].

Abstraction-Based Techniques In [BPG08], the Assume-guarantee-Abstraction Refinement method is proposed for the automation of this compositional reasoning paradigm and applied for Rule 2.1. It is mainly inspired by the CEGAR approach and iteratively finds assumptions that represent the abstraction of the behavior of component B' which concerns its interaction with component B. Therefore, in each iteration, the assumption A satisfies the first premise of the rule and is verified for the second premise. The generated counterexamples are analyzed and those which are spurious serve to strengthen iteratively the overapproximation made by the abstraction.

2.1.2 Assume-guarantee Reasoning for Timed Systems

The above-mentioned research works are focused on the untimed systems. The proposals to handle real-time systems are more limited. In [GJL04, GJP06, GJL10], the authors extended the learning techniques which were first proposed by Angluin for finite automata to the setting of real-time systems. In the field of timed automata, one major obstacle for learning assumptions is that the set of clocks is not known in advance. Regarding this difficulty, the authors restricted their methods for *event-recording-automata* (ERA) [AFH99] where the last occurrence of each action is registered. The three developed algorithms focus on ERA with canonical shapes and which can be perceived as finite automata over a symbolic alphabet.
Another method for learning timed systems was proposed in [VDWW06]. It deals with timed automata containing one clock reset at every transition. There, the generalization to automata with multiple clocks is difficult. In [LAL+14], another method for learning the non circular rule of assume-guarantee method is proposed for the verification of timed systems, focusing also on ERA. It consists in a two-steps flow: first, an untimed assumption is generated in order to ensure the events sequence is correct, then these assumptions are refined so that timing constraints are satisfied.

In [FHK04], an assume-guarantee reasoning approach is proposed for hybrid I/O-automata [LSV03] where a novel rule is proposed on the basis of simulation relations. One advantage of this approach is that circularity is countered by a state-based non blocking condition that can be checked during the computation of the simulation relations.

2.1.3 Limitation

Despite learning techniques may help the automation of the assumptions generation task, one other major prior problem remains, which is finding the right decompositions into subsystems. In fact, systems are usually composed of more than two components. For the application of Rule 2.1 for example, both components B and B' need to be made up of several components. This decomposition task has a decisive impact on the complexity of the verification task and is crucial. In [CAC08] , a study was conducted in order to investigate on finding the best decomposition and assessing the complexity reduction offered by assume-guarantee reasoning. The authors proved that in many cases, the selection of the decompositions did not reduce the memory usage of the verifier in comparison with monolithic verification.

2.2 Contract-Based Reasoning and Interface Theories

The notion of contracts first appeared in the object-oriented programming domain [Mey92] and has been extended to other contexts. Contract-based reasoning aims at defining abstract component specifications precising how it participates in the achievement of a particular requirement in a given environment. When not all the models of the components of the system are completely available, notably at the early design phases, for example when the subsystems are developed by different teams, one is led to think at the level of assumptions and guaranteed properties. Contracts [BCP07, BFM+08] and interfaces appeared as an alternative in such cases. Besides their usability as design constraints to preserve along the development process, contracts may be also used at compositional verification aim.

The theory of interfaces has been proposed to handle incremental design by composition of interfaces. De Alfaro and Henzinger [dAH01a, dAH01b] introduce an interface theory detailing how a component and its environment should interact

through input/output composition. There, the constraint expressed on the inputs represent the assumption whereas the constraint on outputs represent the guarantee. This work inspired the appearance of interface I/O automata proposed by Larsen et al. in [LNW06]. An I/O interface clearly separates through two I/O automata the assumption over the environment and the guarantee offered by the component if this assumption is valid. The separation of the assumption and guarantee offers a better flexibility and re-usability. For instance, it eases the refinement checking in case a new environment refines a previous one.

In [QG08], a methodology to reason about contracts for a system obtained by hierarchical composition of components is proposed. Like interface I/O automata, it separates the assumption about the environment and the guarantee offered by the component whenever this assumption is satisfied by the environment. This methodology offers a *meta-theory* in the sense that it does not propose a new generic design framework, but rather a set of conditions that a contract theory must satisfy in order to apply some specific proofs. There, a special focus is paid to circular reasoning.

Independently from contracts and in relation with interface theory, a number of specification theories have been developed, including abstract specification of modal finite-state transition systems, that is transition systems comprising *must* transitions pointing out required behavior and *may* transitions modeling optional, that is to say allowed but not necessary behavior. Such systems have the particularity of allowing loose specifications with stepwise refinement. Concerning real-time systems, we mention the works in [dAHS02, TWS06, BLPR09, DLL+10]. In [DLL+10], the proposed specification theory comes with the ECDAR branch of UPPAAL [DLL+10] as a tool support.

Specification theories differ from contract-based reasoning in that they don't strictly separate the specification of guarantees from the specification of assumptions. Despite having some similarities and targeting similar objectives towards compositional reasoning, contract and specification theories first progressed independently without a deep investigation of the degree of their complementarity. In [BDH+12], Bauer et al. proposed a generic way to derive a contract framework based on assume-guarantee pairs from a given component-based specification theory supporting specification refinement and composition.

2.3 Summary

In this chapter, we briefly reviewed some major research works conducted in the field of compositional verification. A succinct presentation of the assume-guarantee and contract-based reasoning methodologies was given. We note that the proposals to handle compositional reasoning for real-time systems are rather limited.

Our compositional method is based on the deductive approach and differs from the mentioned frameworks in that it is not restricted to systems with binary interactions but is directly applicable to systems containing multi-partly interactions. Besides, in some way, it needs only *guarantees* about the components offered by the local invariants, whereas assumptions about the environment are substituted by the generation of interaction invariants along with another type of invariant which reflects the time-synchronization between the different components. Furthermore, in our framework, the computation of the global invariant is independent from the property of interest.

Chapter 3

Background

Contents

The design technique has a significant influence on the verification cost of a given system. The aim of verification is to decide whether a system meets some requirements and to provide a diagnostic when it is not the case. However, with the

growing complexity of nowadays systems, the correction can be very expensive. The clearer is the design of the system, the easier and the less costly is its verification. Component-based design techniques are proposed to tackle the complexity of the design issue. The main idea consists in building complex systems by assembling less complex entities. Therefore, the behavior of the system is split into logical units encapsulating separate behaviors. This technique allows to construct evolving and flexible systems where the addition, the removal or the modification of a component does not affect the inner behavior of the others. Composition operation allows to construct a system from a set of components. A component has a well defined interface to interact with the other components while the composition provides the glue operators which link the components with each other. Beyond the evolutivity and flexibility, component-based design offers the possibility to guarantee the correctness of the system by assembling components with known properties. For the design framework to be constructive, it should meet two conditions: composability and compositionality. The former requirement implies that each component preserves its properties after composition with the other components while compositionality allows to infer global properties of the system from the properties of the sub-components.
In the first part of this chapter, we present BIP, a component-based design framework which separates clearly the behaviors from the glue operations. In the second part, we describe a fully compositional method which was proposed for the verification of untimed systems modeled in the BIP language.

3.1 BIP framework

Component-based design consists in building complex systems from a set of components described by their behaviors and a set of glue operations modeling the coordination between the different components. BIP (Behavior-Interactions-Priorities) is a framework for component-based design where the coordination between components is introduced by two kind of glue operations: interactions and priorities. Its architecture is depicted in Figure 3.1 and consists of three layers:

- Behavior: The behavior of a component is described by a labeled transition system where the transitions are defined by an action name, a guard, that is a condition for the transition to be enabled and an update function.
- Interactions: The interactions restrain the global behavior of the composition. They impose the possible synchronizations between the actions of the

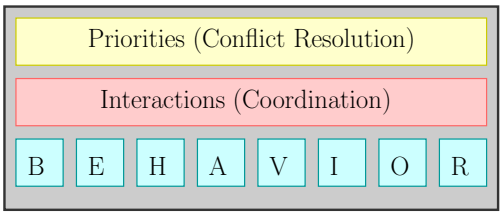


Figure 3.1: BIP layers

different components.

- Priorities: They restrict the global behavior by filtering among a set of possible interactions at a given time. They are useful to strengthen state invariants and restrain non-determinism.

In the following, we give a formal description of the above layers.

3.1.1 Atomic Components

In BIP, the behavior of an atomic component is described by a labeled transition system enriched with data variables. It possesses a set of locations, a set of transitions, and a set of actions serving for the synchronization with the other components.

Definition 3.1 (Labeled transition system)**.** *A labeled transition system is a quadruple* (L, A, T) *where* L *is a set of locations,* A *is a set of actions,* $T \subseteq L \times A \times L$ *is a set of transitions, each labeled by an action.*

For any action a and pair of locations l and l' , we write $l \xrightarrow{a} l'$ iff $(l, a, l') \in T$

Definition 3.2 (Atomic component)**.**
An atomic component $B = (L, A, T, \mathcal{D}, \{g_t\}_{t\in T}, \{f_t\}_{t\in T}, Init)$ *is a transition system enriched with data variables as follows:*

- (L, A, T) *is a labeled transition system.*
  - $L = \{l_1, l_2, \ldots l_n\}$ *is a set of locations.*
  - $A = \{a_1, a_2, \ldots a_m\}$ *is a set of actions.*
  - $T = L \times A \times L$ *is a set of transitions.*
- $Init = (l_0, \zeta_0)$ *is an initial state. It is defined by a location* l_0 *and a predicate* ζ_0 *characterizing the initial values of the data variables.*
- $\mathcal{D} = \{d_1, d_2, \ldots d_k\}$ *is a set of data variables and*

- *for each $t \in T$, we designate by g_t a guard relative to t, that is a predicate on $\mathcal{D}$ that should be satisfied for the transition t to fire, and by $f_t(\mathcal{D}, \mathcal{D}')$ an update relation that is, a predicate on $\mathcal{D}$ (current) and $\mathcal{D}'$ (next) state variables.*

A transition in T has the form $t = (l, a, g_t, f_t, l')$ where l is the source location, l' is the destination location, a is the action from which the component is synchronized to one or more interactions. An action can be also internal, meaning that it is not synchronized with any action from other components. A transition can fire only if the guard g_t is satisfied for the current valuation of data variables while f_t is an update step consisting of local state transformation. It can be represented by (l, a, l') in case where g_t is equal to **true** and where there is no internal update f_t. An action is enabled if one of its transitions is enabled and is disabled if all of its transitions are disabled. A transition (l, a, g_t, f_t, l') is enabled if the component is in the source location l and the guard g_t is true for the current data valuations. At the opposite, it is disabled if the component is not at location l or the guard g_t is false. The behavior of an atomic component is expressed by a labeled transition system with transitions of the from $(l, d) \xrightarrow{a} (l', d')$, where l and l' are control locations of the component and d and d' are the valuations of the data variables at locations l and l' respectively. The move $(l, d) \xrightarrow{a} (l', d')$ is possible if there is a transition $t = (l, a, g_t, f_t, l')$ such that $g_t(d) \equiv$ **true** and $f_t(d, d') \equiv$ **true**.

Example 3.1 (An atomic component)**.** Graphically, a component is presented by a box containing its behavior. The circles depict the locations, the double circle designates the initial location, the arrows show the transitions between the source and destination locations, and the small rectangles on the borders of the component box show the actions serving to connect the component with other components. Each transition is labeled with the name of its triggering action. Figure 3.2 depicts a component having the initial location lc_1. Its initial state *Init* is defined by $Init = (lc_1, \theta = 100)$.

The semantics of an atomic component is defined by the following transition system.

Definition 3.3 (Semantics of an atomic component)**.**
The semantics of component $B = (L, A, T, \mathcal{D}, \{g_t\}_{t \in T}, \{f_t\}_{t \in T}, Init)$ is a transition system $(Q, A, T_s, Init)$ where

- *$Q = L \times \mathbf{D}$ is a set of states where $\mathbf{D}$ is the set of valuations of variables in D.*
- *T_s is a set of transitions $((l, v), a, (l', v'))$ such that $g_t(v) \wedge f_t(v, v')$ for some $t = (l, a, l') \in T$. To express that $((l, v), a, (l', v')) \in T_s$, we write $((l, v) \xrightarrow{a} (l', v'))$.*

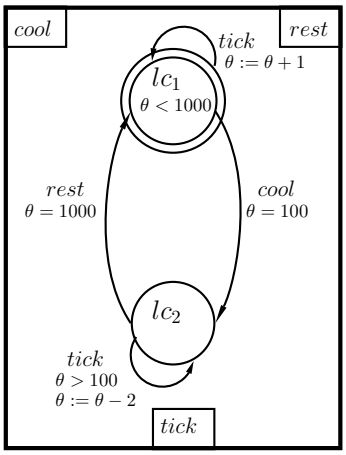


Figure 3.2: An atomic component

Definition 3.4 (Reachable states of a component)**.**
Given a component B having the initial state Init, a state s is reachable if there exists an execution sequence $Init \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$ *such that* $s_n = s$.

3.1.2 Interactions and Parallel Composition

In the BIP framework, the coordination between the components is insured by the use of interactions. An interaction serves to synchronize a set of actions from the different components. The synchronization is possible when each of the interaction's actions is ready to be executed through a transition of the component to which it belongs.

Definition 3.5 (Interaction)**.**
Given a set of components $B_i = (L_i, A_i, T_i, \mathcal{D}_i, \{g_{t_i}\}_{t_i \in T_i}, \{f_{t_i}\}_{t \in T_i}, Init_i)$, $i = 1 \ldots n$ *an interaction is a subset* $\alpha \subseteq \cup_{i \in I} A_i$, *with* $I \subseteq \{1 \ldots n\}$ *such that* $\forall i \in I$, $|\alpha \cap A_i| \leq 1$

We denote by $involved(\alpha)$, the set of components which have one action participating in interaction α. Formally,

$$involved(\alpha) = \{B_i | A_i \cap \alpha \neq \emptyset\}$$

For readability, we make use of the notation $(a_1|a_2|\ldots|a_k)$ for interactions instead of $\{a_i\}_{i=1\ldots k}$. The interaction model is specified by a set of interactions $\gamma \subseteq 2^A$, where $A = \bigcup_{1\ldots n} A_i$. For a subset λ of interactions, we denote by $Act(\lambda)$ the set of actions involved in the interactions belonging to λ.

During the execution, an interaction can be *enabled* or *disabled*. An interaction is enabled iff all of its actions are enabled. Contrarily, it is disabled iff at least

one of its actions is disabled. Two interactions are conflicting if they share one action or more. In this case, one and only one of them can be executed. In the BIP framework, the data transfer on interactions is possible. For an interaction α, we denote by G_α (a boolean condition) the guard and by F_α the data transfer function allowing the update of data variables during the interaction execution. Both operate on the variables existing in the participating components.
The parallel composition of a set of components is defined as follows:
Definition 3.6 (Parallel composition)**.**
Given n components $B_i = (L_i, A_i, T_i, \mathcal{D}_i, \{g_{t_i}\}_{t_i \in T_i}, \{f_{t_i}\}_{t \in T_i}, Init_i)$*, where the sets* A_i *of actions are disjoint, and a set of interactions* γ*, we define their parallel composition* $\|_\gamma B_i$ *as the component* $(L, \gamma, T_\gamma, \mathcal{D}, \{g\}_{t \in T_\gamma}, \{f\}_{t \in T_\gamma}, Init)$*, where*

- $L = L_1 \times L_2 \ldots L_n$ *is the set of global locations.*
- $\mathcal{D} = \cup_i \mathcal{D}_i$ *is the set of data variables.*
- *The set of global transitions obtained by synchronization of sets of transitions* $\{t_i = (l_i, a_i, g_{t_i}, f_{t_i}, l_i') \in T_i\}_{i \in I}$ *for an arbitrary* $I \subseteq \{1, \ldots, n\}$*, is defined by*

$$T_\gamma = \left\{ (\bar{l}, \alpha, g, f, \bar{l}') \;\middle|\; \begin{array}{l} \bar{l} = (l_1, ..., l_n) \in L, \;\; \bar{l}' = (l_1', ..., l_n') \in L \\ \alpha = \{a_i\}_{i \in I} \in \gamma, \;\; \forall i \in I. (l_i, a_i, g_{t_i}, f_{t_i}, l_i') \in T_i, \;\; \forall i \notin I. l_i = l_i' \\ \qquad f \equiv F_\alpha \circ \bigcirc_{i \in I} f_{t_i}, g \equiv G_\alpha \circ \bigcirc_{i \in I} g_{t_i} \end{array} \right\}$$

- *Init is the initial state of the obtained composite system. It gives the initial location where each of the composing components initially is. It includes also the conjunction of the initial constraints on data variables of all of them. Formally, if for every component* B_i*,* $Init_i = (l_{0i}, \zeta_{0i})$*, then* $Init = ((l_{01}, l_{02}, \ldots, l_{0n}), \bigwedge_i \zeta_{0i})$*.*

The behavior resulting from this parallel composition can thus execute a transition relative to $\alpha \in \gamma$ iff for every $i \in I$, the action $\alpha \cap A_i$ is enabled in component B_i. The locations of the components which do not participate in the transition remain unchanged.
Example 3.2 (Parallel composition)**.** Figure 3.3 depicts the parallel composition of two components B_1 and B_2 by a set of interactions $\gamma = \{(a), (b|c)\}$. Since action b of component B_1 and action c of component B_2 are synchronized through interaction $b|c$, each one of them can be executed only simultaneously with the other. The system resulting from the parallel composition of B_1 and B_2 have two possible actions bc and a.

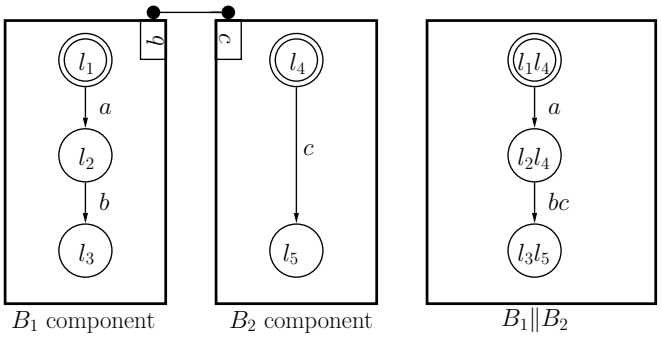


Figure 3.3: Parallel composition of two components

3.1.3 Priority Rules

Priorities serve to reduce the non-determinism. They allow to select interactions among the enabled interactions at the current state of the system.

Definition 3.7 (Priority)**.**
A priority is a relation $\prec \subseteq \gamma \times L \times \gamma$, where L is the global set of locations and γ is the set of interactions. The formula $a \prec_l a'$ expresses that interaction a has priority on interaction a' at location $l \in L$. For all l, the relation $\prec_l$ represents a partial order on γ.

Intuitively, priorities of a composite system are ordered pairs of interactions. For each pair, when both of the interactions are enabled, only the interaction with the highest order of priority is allowed to fire.

3.1.4 An Example of BIP System

In the following, we illustrate the application of the BIP component-based system design on the Temperature Control system [ACH+95].

Example 3.3 (The Temperature Control system modeled in BIP)**.** This system is aimed to control the temperature in a nuclear reactor and to keep it within some bounds. In practice, this system is composed of a controller component interacting with a set of rods. We show first the system for two rods. Those latter components are used to cool the reactor when a maximum temperature 1000° of the reactor is reached in which case one of them starts cooling it until the minimum temperature 100° is reached. At this time, the operating rod rests and the temperature within the reactor rises again. When no rod is ready to refrigerate the reactor when necessary, the system is completely shut down. As in the untimed BIP framework, the time is not handled, it is substituted in this example by the *tick* transitions. The rods Rod_1 and Rod_2 are identical up to locations and variable

names. They have the same structure, two locations and four transitions. Among them, there are two loop transitions triggered by *tick* actions and synchronized with the Controller *tick* action. The interaction structure is modeled by $\gamma = \{cool|cool_1, cool|cool_2, heat|rest_1, heat|rest_2, tick|tick_1|tick_2\}$. While the actions in the interaction $tick|tick_1|tick_2$ are strongly synchronized, the interactions $heat|rest_1$ and $heat|rest_2$ are conflicting on action *heat*. Only one of them can be executed at any given time. The initial state of the system is defined by $Init = ((l_{11}, lc_1, l_{12}), \theta = 100 \wedge t_1 = 3600 \wedge t_2 = 3600)$.
The Controller component contains two locations: lc_1 location meaning that the reactor is in the heating position and lc_2 location, reflecting that the temperature is being decreased by use of one of the rods. The variable θ represents the temperature which decreases by two units at lc_2 when transition *tick* fires and increases by one unit at lc_1 location when when transition *tick* fires. The set of transitions relative to this component are the following:

- the transition t_1 triggered by *tick* action: $t_1 = (lc_1, tick, g_{t_1} = (\theta < 1000), f_{t_1} = (\theta := \theta + 1), lc_1)$. t_1 is a loop transition, meaning that the component remains at the same location lc_1 after its execution.

- the transition t_2 triggered by *cool* action: $t_2 = (lc_1, cool, g_{t_2} = (\theta = 1000), lc_2)$. There is no internal computation, proper to this transition. This transition is synchronized through action *cool* with either action $cool_1$ from Rod_1 or $cool_2$ from Rod_2. If none of them is at l_{11} (resp. l_{12} location), and has variable t_3 (resp. t_2) with a value bigger or equal to $3600°$, then the system is deadlocked, meaning that it is unable to execute any action. In this case, it should be shut down.

- the loop transition t_3 triggered by *tick* action: $t_3 = (lc_2, tick, g_{t_3} = (\theta > 100), f_{t_3}(\theta := \theta - 2), lc_2)$. Every execution of t_3 decreases the temperature with 2 degrees. The action *tick* is synchronized with $tick_1$ and $tick_2$ from Rod_1 and Rod_2 respectively.

- the transition t_4 triggered by *heat* action: $t_4 = (lc_2, tick, g_{t_4} = (\theta = 100), lc_1)$. It synchronizes with either action $rest_1$ of Rod_1 or $rest_2$ of Rod_2.

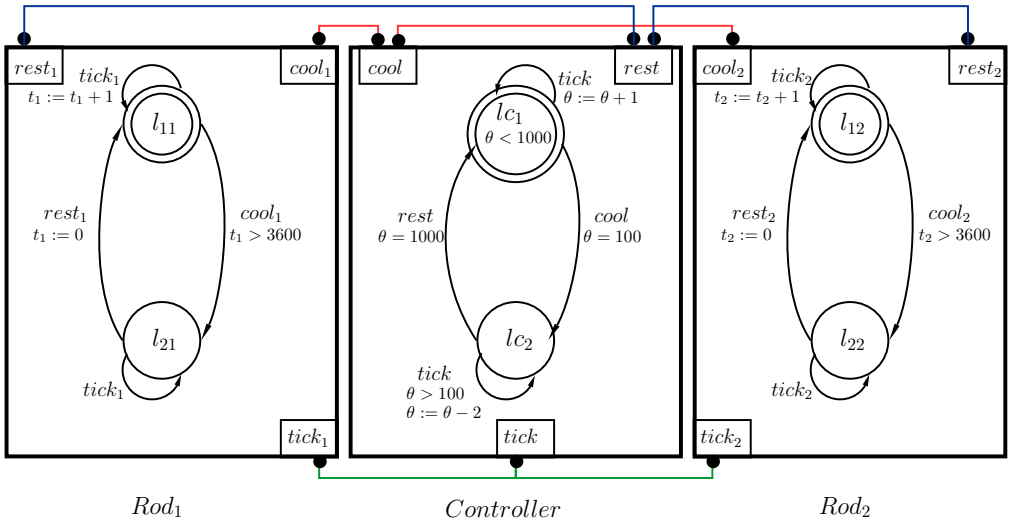


Figure 3.4: Temperature Control system modeled in BIP

3.2 Properties of BIP Systems

3.2.1 Invariants

A state predicate I is an invariant of a component B if it is satisfied by all the reachable states, meaning that every state reached during the computation of B satisfies it.

The *post* predicate transformer computes the state successors of the atomic components which are represented symbolically by state predicates.

Definition 3.8 (Post predicate transformer with respect to transition)**.** *Given a component $B = (L, A, T, \mathcal{D}, \{g_t\}_{t\in T}, \{f_t\}_{t\in T}, Init)$, and a state predicate φ on the variables in $\mathcal{D}$, the post predicate transformer with respect to a transition $t = (l, a, g_t, f_t, l')$ is defined as follows:*

$$post_t(\varphi)(\mathcal{D}) \equiv \exists \mathcal{D}'.g_t(\mathcal{D}') \wedge f_t(\mathcal{D}', \mathcal{D}) \wedge \varphi(\mathcal{D}')$$

We refer by $at(l)$ to the boolean variable expressing that a given component is at l location.

Example 3.4 (Post predicate computation w.r.t transition)**.** Figure 3.5 depicts the propagation of the predicate $\varphi(d) \equiv (d \geq 0)$ through transition t having the guard $g_t = (d > 2)$ and the update function $f_t = (d := d + 1)$. The post condition of φ with respect to the transition t is $\varphi'(d) \equiv \exists d'.(d' > 2) \wedge (d := d' + 1) = (d > 3)$.

For a component $B = (L, A, T, \mathcal{D}, \{g_t\}_{t\in T}, \{f_t\}_{t\in T}, Init)$, given a state predicate $\phi = \bigvee_{l\in L} at(l) \wedge \varphi_l$, the post predicate with respect to the transition system is

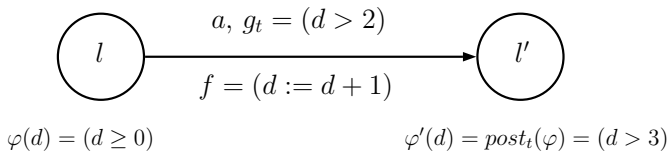


Figure 3.5: Example of post-predicate computation

defined as

$$post(\phi) \equiv \bigvee_{l \in l} at(l) \wedge (\bigvee_{t=(l',a,l)} post_t(\varphi_{l'}))$$

The computation of $post(\phi)$ can be performed by forward propagation following the control locations in ϕ.

Definition 3.9 (Inductive invariant)**.** *For a component B, a state predicate ϕ is*

- *an inductive invariant iff $post(\phi) \Rightarrow \phi$ and ϕ is satisfied at the initial state.*
- *an invariant iff there is an inductive invariant ϕ_{aux} that implies it: $\phi_{aux} \Rightarrow \phi$.*

The set of reachable states of a given component forms an inductive invariant. Every over-approximation of this set is therefore an invariant of the component.

3.2.2 Deadlocks

3.2.2.1 Local Deadlocks

A component is locally deadlock-free if one of its transitions can be executed.

Definition 3.10 (Local Deadlock-freedom)**.**
Given a component $B = (L, A, T, \mathcal{D}, \{g_t\}_{t \in T}, \{f_t\}_{t \in T}, Init)$, the state predicate characterizing the set of deadlock-free states is expressed as

$$DFS \equiv \bigvee_{l \in L} \bigvee_{t \in l^\bullet} enabled(t)$$

where $l^\bullet$ contains the transitions having l for source location and $enabled(t)$ is the predicate expressing the enabledness condition for transition t.
Formally, $enabled(t) \equiv at(l) \wedge g_t$. This means that for the transition t to fire, the component should be at its source location l and the guard g_t should be satisfied.

The DFS implies that a component is deadlock-free if, whichever is the location that it reaches, there exists an outgoing transition that is enabled.

3.2.2.2 Global Deadlocks

A system is deadlocked if in the current state, no interaction is enabled. We first introduce the enabledness condition for an interaction.

Definition 3.11 (Interaction enabledness)**.** *An interaction α is enabled if all of its actions are enabled, that is for each of them, at least one of the associated transitions is enabled.*

$$enabled(\alpha) \equiv \bigwedge_{a \in \alpha} enabled(a)$$

where $enabled(a) \equiv \bigwedge_{t \in trans(a)} enabled(t)$ and $trans(a)$ is the set of transitions triggered by action a. In general, this set does not contain a unique transition because a component can have many transitions labeled with the same action a.

The predicate DIS expresses the set of states where the system is deadlocked.

Definition 3.12 (Deadlock States Set)**.** *The predicate DIS characterizing the set of all the states from which all the interactions are disabled is defined as*

$$DIS \equiv \bigwedge_{\alpha \in \gamma} \neg enabled(\alpha)$$

3.3 Compositional Verification

In [BBSN08], the set of reachable states is over-approximated by invariants. Mainly two types of invariants were proposed: local invariants characterizing the components and an *interaction invariant* characterizing their coordination.

- *Component invariants*: they are local invariants characterizing the internal behavior of the components independently from their interactions with each other. The invariant of a given component is thus preserved even if the other components of the system are changed.
- *Interactions invariants*: the interactions synchronize actions of different components. They restrain the possible global locations of the system. Interaction invariants aim to capture constraints on the possible combinations of the locations of the different components. They can either be computed from Boolean Behavioral Constraints which represent sets of implications induced from the interaction structure and the local behaviors as shown in [BBSN08], or as proposed in [BBBL13], they can be represented by sets of linear equations.

By taking the conjunction of the above invariants for $Reach_{app}(S)$, the verification rule becomes:

$$\frac{\forall_i B_i \models \Box CI(B_i), \quad II(\gamma), \quad \bigwedge_i CI(B_i) \wedge II(\gamma) \Rightarrow \Psi}{\|_\gamma B_i \models \Box \Psi} \text{ (D-Finder VR)}$$

In the above rule, $II(\gamma)$ is the interaction invariant expressing constraints on the global locations induced from the interaction structure γ. The component invariant of B_i is formulated in $CI(B_i)$. If the computed invariant $(\bigwedge_i CI(B_i) \wedge II(\gamma))$ implies the safety property Ψ, then the system $\|_\gamma B_i$ obtained by the parallel composition of the components B_i satisfies it. The verification of the invariance of Ψ is therefore done by checking the satisfiability of $(\bigwedge_i CI(B_i) \wedge II(\gamma) \wedge \neg\Psi)$.

3.3.1 D-Finder

The above rule was implemented in the D-Finder tool [BBNS09] designed to verify safety properties for component-based systems described in the BIP language, with a focus on deadlock-freedom. It takes as input a system modeled in the BIP language and progressively finds and eliminates probable deadlocked configurations. D-Finder consists of a set of interconnected modules and performs basically following these steps:

1. Construction of the predicate DIS characterizing the deadlock states set.
2. Iterative computation of increasingly stronger component invariants.
3. The computation of interaction invariants by collaborating with the CUDD package or a SAT-solver.
4. Satisfiability checking of $\bigwedge_i CI(B_i) \wedge II(\gamma) \wedge DIS$. In case of systems without data, the CUDD package is used, else a Sat-Solver.

In the following, we give a detailed presentation of the computation of the proposed invariants.

3.3.2 Component Invariants

Component invariants are local invariants over-approximating the set of reachable states of the components separately from each other. In [BBSN08], they are generated the by use of the *post* predicate transformer computing the state successors of the atomic components which are represented symbolically by state predicates.

One method for generating inductive invariants for components is defined in the following proposition.

Proposition 3.1 (Computation of inductive component invariant)**.** *Given a component B with initial state Init* $= (l_0, \zeta_0)$*, the following iteration is a sequence of increasingly stronger inductive invariants.*

$$\phi_0 \equiv \textbf{true}, \quad \phi_{i+1} \equiv (at(l_0) \wedge \zeta_0) \vee post(\phi_i)$$

One technique for generating such invariants consists in iterating until finding strong enough invariants implying the desired safety property. The precise computation of this sequence requires quantifier elimination. As an alternative, it was proposed to compute rather an over-approximation of the *post* predicate based on the predicates analysis. A syntactic technique was proposed to over-approximate the post predicate transformer $post_t$ with respect to a given transition t. The main idea lays upon finding predicates $post_t^a$ which are not affected by the transfer function f_t. They are proved to be invariants of the component since they are implied by the inductive invariants $post_t$.

3.3.3 Interaction Invariants

While component invariants are used to reflect the behaviors of the different components in isolation from each other, the interaction invariant is intended to capture the global locations induced from the coordination between them. Interactions are used to constrain the global locations of the system. An interaction gathers a set of actions which can occur simultaneously. Therefore, they engender strong constraints on the moves of the components.

Figure 3.6 illustrates a system composed of two components B_1 and B_2 strongly synchronized through interactions $a_1|a_2$ and $b_1|b_2$. They have the initial locations l_0 and l_2 and the component invariants $l_0 \vee l_1$ and $l_2 \vee l_3$ respectively. The conjunction of these component invariants does not track the strong synchronization induced by the interaction structure and is very likely to not detect invariance safety properties. The interaction invariant aims to extract constraints on the global location of the system based on the interaction structure.

In the following of this section, we first show the method that has been proposed for the computation of interaction invariant for systems without data variables. For infinite systems, that is systems with data variables, an abstraction step is first performed before the application of the basic method for finite state systems.

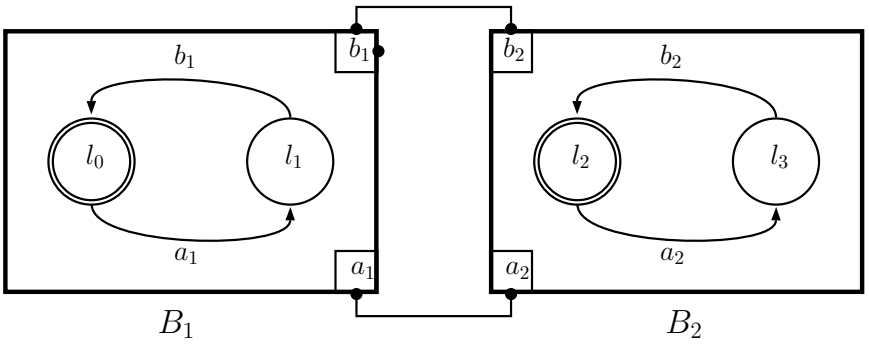


Figure 3.6: A BIP system with two components

3.3.3.1 Finite State Systems

If we consider a set of components $\{B_1, B_2, \ldots B_n\}$ which are synchronized through a set of interactions γ, then the interaction invariant is a predicate in the disjunctive form $\bigvee_{l \in L_\gamma} l$ where $L_\gamma \subseteq \bigcup_{1 \ldots n} L_i$. For any control location in L_γ, there exists for the execution of any interaction in γ, a control location in L_γ which is reached. Intuitively, every L_γ is a set of locations of components such that for any location $l \in L_\gamma$, the successor of l does also belong to L_γ. This predicate is an invariant if it is initially true, that is it has at least an initial location of an atomic component B_i. The interaction invariant is computed either by solving Boolean Behavioral Constraints (BBCs) characterizing the successor locations of every location or Fixed-point computation. In [BBBL13], another method was proposed to compute incrementally linear interaction invariants.

In the following, we describe briefly the method based on solving BBCs.

Definition 3.13 (Forward interaction set)**.** *Given the parallel composition $\|_\gamma B_i$ of a set of components, for each location $l \in \bigcup_{1 \ldots n} L_i$, the forward interaction set relative to l is defined as*

$$l^\bullet = \{\{t_i\}_{i \in I} | \forall i.t_i \in T_i \wedge (\exists i.{}^\bullet t_i = l) \wedge \{action(t_i)\}_{i \in I} \in \gamma\}$$

For a set of locations L, the forward interactions set is $L^\bullet = \bigcup_{l \in L} l^\bullet$.

In the above definition, ${}^\bullet t$ (resp. $t^\bullet$) denotes the set of source (resp. destination) locations of the transition t. That is, for a location l, $l^\bullet$ contains the set of transitions belonging to an interaction involving a transition t_i having l for source location.

Similarly, the backward interactions set is defined as ${}^\bullet L = \bigcup_{l \in L} {}^\bullet l$ where $l^\bullet = \{\{t_i\}_{i \in I} | \forall i.t_i \in T_i \wedge (\exists i.t_i^\bullet = l) \wedge \{action(t_i)\}_{i \in I} \in \gamma\}$.

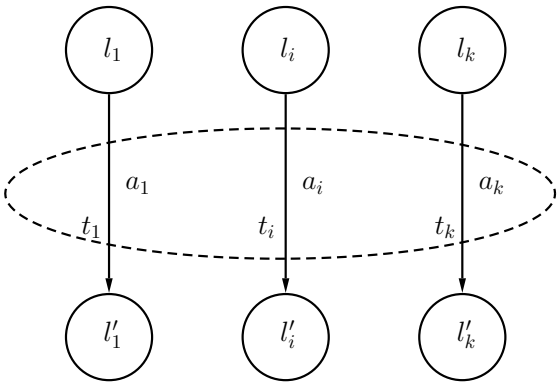


Figure 3.7: Forward interaction sets

Example 3.5 (Forward interaction set)**.** For example, in Figure 3.7, the set $\{t_1, \ldots, t_i, \ldots t_k\}$ belongs to the forward interaction set $l_1^\bullet$ of location l_1. It does also belong to the forward interaction sets of locations $l_2, \ldots l_i, \ldots l_k$.

Those sets can be perceived as transitions in $1-$safe Petri nets where the notion of traps is defined as follows:

Definition 3.14 (Traps)**.** *Given a system S defined as the parallel composition $\|_\gamma B_i$, where $B_i = (L_i, A_i, T_i, \mathcal{D}_i, \{g_{t_i}\}_{t\in T}, \{f_{t_i}\}_{t\in T_i})$, a trap is a set L of locations such that $L^\bullet \subseteq {}^\bullet L$.*

If the initial location of the system has a control location in some trap then all of its successors have a control location belonging to the trap. If the set of locations $L \subseteq \bigcup_{1\ldots n} L_i$ is a trap containing the initial location of one of the system components, then $\bigvee_{l\in L} at(l)$ is an invariant of S. The traps can be computed as solutions of the following system of implications:

If a boolean valuation $\mathbf{v} : \bigcup_{i=1\ldots n} L_i \to \mathbb{B}$ satisfies the below set of applications, then the set $\{l \in \bigcup_{i=1\ldots n} L_i | \mathbf{v}(l) = \mathbf{true}\}$ is a trap. The required implications for $l \in \bigcup_{i=1\ldots n} L_i$ are defined as follows:

$$\mathbf{v}(l) \Rightarrow \bigwedge_{\{t_i\}_{i\in I}\in l^\bullet} \Big(\bigvee_{l'\in\{t_i^\bullet\}_{i\in I}} \mathbf{v}(\mathbf{l}')\Big)$$

This allows to generate the traps in an enumerative manner. This requires the use of a SAT-solver in order to obtain the minimal solutions of the above system. It has been demonstrated in [YW99] that this traps computation method is NP-complete.

Example 3.6 (Traps and interaction invariant)**.** The set of of minimal traps for the example given in Figure 3.6 are:

$$L_1 = \{l_0, l_1\}, L_2 = \{l_1, l_2\}, L_3 = \{l_0, l_3\}, L_4 = \{l_2, l_3\}$$

The resulting interaction invariant is $II = (l_1 \vee l_2) \wedge (l_0 \vee l_3)$

A detailed presentation of interaction invariant computation methods is shown in [Ngu10].

3.3.3.2 Infinite State Systems

In case of infinite systems, that is to say systems with data variables, an abstraction step is first performed. It aims at computing an abstract component B_i^α from each component B_i of the system through an abstraction function α. The interaction invariant of the composite system $S = \|_\gamma B_i$ is deduced from the interaction invariant of $\|_\gamma B_i^\alpha$. In fact, the abstract component B_i^α is derived from the concrete one with respect to an invariant of B_i following the disjunctive form and with respect to its abstraction function in such a way that the two following conditions are guaranteed

- The system S^α resulting from the parallel composition of the abstract components simulates S.
- If ϕ^α is an invariant of S^α then $\alpha^{-1}(\phi^\alpha)$ is an invariant of S. It follows that it is possible to obtain the interactions invariant of the concrete system S from the traps corresponding to the interactions invariant of the abstract system S^α.

3.3.3.3 Dealing with Data Transfer on Interactions

The explained method does not handle the data transfer on interactions. The data transfer allows to update the data variable values during the execution of the interactions, hence their values do not depend only on the behavior of the component to which they belong. The above introduced method handles only pure interactions, that is interactions without transfer of data variables. The absence of data transfer on interactions is a hurdle in practice and its ignorance may cause the global invariant not being tight enough to detect some properties. To preserve the compositional verification reasoning and allow data transfer consideration for stronger invariant computation at the same time, a two-fold scheme was proposed in [Ngu10]. It lays upon these two major techniques:

- The changes of the data variables in interactions are projected to the components to which the data belong. This allows to transfer the data updates on interactions to the level of the components' transitions. This influences the computation of the post-predicate transformers over transitions and by consequence the local components' invariants.
- For interaction invariant, the so-called *interaction component* is introduced to mimic the data transfer over the set γ of interactions. The interaction structure is subsequently updated by connecting the actions of the interaction component. This results in an abstract system without data variables transfer on interactions.

After performing the above two steps, consisting in the modification of the components transitions and adding and connecting a new interaction component, the compositional verification method can be applied on the obtained abstract system allowing the global invariant to capture the effect of data transfer.

3.4 Summary

Component-based design aims to alleviate the complexity issue while building complex system by assembling logical entities with less complex features. In this chapter, we presented BIP, a modeling framework where the system construction follows three layers: Behavior, Interactions and Priorities. The Behavior layer includes the behaviors of the different components modeled mainly by transition systems which can be enriched with data variables. The Interactions layer expresses the strong synchronization between the different components and the Priorities layer allows to restrain the non-determinism by filtering among enabled interactions at a given location.
In the second part of this chapter, we recalled a compositional verification method proposed for such systems. It is based on a fully automatic and compositional method for the computation of an invariant over-approximating the set of reachable states.
In the presented design framework, the time modeling is not supported. Besides, in general, a direct application of the proposed verification rule to timed systems would not be strong enough to capture the desired properties. In fact, in a timing setting, the clocks of the different components do all advance at a common rate. This induces time-synchronization relations which cannot be captured by the above invariants.

In the next chapter, we present a framework for modeling real-time systems which can be perceived as an extension of the above framework where components are modeled mainly as timed automata.

Chapter 4

Timed Component-Based Systems

“ We may say a thing is at rest when it has not changed its position between now and then, but there is no 'then' in 'now', so there is no being at rest. Both motion and rest, then, must necessarily occupy time. ”

Aristotle , *Reply to Zeno's paradoxes*, 350 BC

Contents

Several models have been proposed to describe the behavior of a timed system. Logic theories may serve to model system description and specifications under the same language: the system behavior is expressed by axioms and inference rules while the specifications are represented as theorems, that is formula that can be built through repetitive inference rules. Modecharts [JM94] represent the first timed extension of statecharts [Har87] which allow the description of concurrent systems graphically, while the *Timed unified modeling language* is proposed as the real-time extension of UML [Dou98]. Conversion schemes have proposed to translate UML systems into other frameworks in order to allow the reuse of the already proposed verification methods or tools [FHD+99, KMR02, OGO06], focusing mainly on systems of timed automata.

Timed automata were first proposed by Alur and Dill [AD94] and they represent a prominent framework for the modeling of timed systems. An expressive model having some common features with timed automata is Time Petri-Nets [Ram74, BM83]. Despite the two models were growing independently from each other at the beginning, a significant interest was dedicated to their mutual comparison (e.g [Srb08]), sometimes with a focus on their expressiveness [BCH+05]. Since the tools handling timed automata seem to be more numerous and developed, translation approaches from particular Time Petri Nets were proposed (e.g [HKSLT02]). This offers the reuse of cross techniques and tools which were successful on timed automata.
In this chapter, we present a constructive component-based framework for real-time systems where components are mainly timed automata.

4.1 Time Formalism

The theory of timed automata was proposed to support automated reasoning about the real-time aspect of systems interacting with physical processes. It handles dense-time models where the time values for event occurrences may be real numbers. A timed automaton is mainly a finite-state Büchi automaton extended with real-valued variables called the *clocks*. Before presenting the syntax and semantics of timed systems, we give an overview of time-related notions.

Clock

We consider real-time dense-time models where time is modeled by a set of real-valued positive variables, the clocks. In a given system, all the clocks increase synchronously with the same rate. The only operation allowed is clock *reset*. Therefore, each clock variable records the amount of time elapsing since its latest reset.

Definition 4.1 (Clock valuation). *For a finite set of clocks $\mathcal{X}$, an $\mathcal{X}$-valuation is a function $\mathbf{v} : \mathcal{X} \to \mathbb{R}_{\geq 0}$ assigning to each clock x a positive real value $\mathbf{v}(x)$.*

The set of all valuations is $\mathbb{R}^{\mathcal{X}}_{\geq 0}$.

Definition 4.2 (Clock Constraint). *A clock constraint is defined by the grammar:*

$$C ::= \mathbf{true} \mid x\#ct \mid x - y\#ct \mid C \wedge C$$

where x, y are clocks in $\mathcal{X}$, $\# \in \{<, \leq, =, \geq, >\}$ and $ct \in \mathbb{Z}$.
A clock constraint is atomic if it is defined over one clock or two clocks, thus following this grammar:

$$C ::= \mathbf{true} \mid x\#ct \mid x - y\#ct$$

Given a clock constraint C and an *$\mathcal{X}$-valuation* $\mathbf{v}$, $\mathbf{v} \models C$ denotes the evaluation of C in $\mathbf{v}$. It expresses that the valuation satisfies C. The expression $\mathbf{v} \not\models C$ is used in the opposite case, that is when valuation $\mathbf{v}$ does not satisfy C.

Definition 4.3 (Clock Reset). *Given a clock x in $\mathcal{X}$, the valuation $\mathbf{v}\,[x := 0]$ is obtained from $\mathbf{v}$ by setting the clock x to 0 and keeping the rest of clocks unchanged.*

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

For a subset r of $\mathcal{X}$, the valuation $\mathbf{v}\,[r]$ is obtained by setting all clocks of r to 0 and keeping the rest of clocks unchanged.

$$\mathbf{v}\,[r]\,(x) = \begin{cases} 0 \text{ if } x \in r \\ \mathbf{v}(x) \text{ else} \end{cases}$$

Definition 4.4 (Time Delay). *For a real value δ, the notation $\mathbf{v} + \delta$ defines a new valuation $\mathbf{v}'$ where every clock x in $\mathcal{X}$ is delayed by δ value. That is $\mathbf{v}'(x) = \mathbf{v}(x) + \delta$.*

4.1.1 Zones

In the presence of infinite sets of valuations, the use of the so-called 'zone' allows a coarser representation.

Definition 4.5 (Zone)**.** *A zone corresponds to the set of clock valuations that satisfy a clock constraint. A zone is convex by definition.*

Operations on Zones

Several operations on zones are defined. They express the time elapsing, the reset of clocks, the conjunction of zones, etc.

Reset Given a zone ζ and a set of clock reset operations r, the operations $\zeta[r]$ and $[r]\zeta$ are defined as follows:

$$\zeta[r] = \{\mathbf{v}[r] \mid \mathbf{v} \in \zeta\}$$

$$[r]\zeta = \{\mathbf{v} \mid \mathbf{v}[r] \in \zeta\}$$

Intuitively, $\zeta[r]$ contains all valuations resulting from the execution of reset operations in r. The notation $[r]\zeta$ defines the dual operator. It represents the zone which, after resetting clocks in r, yields valuations in ζ.

Conjunction Given a zone ζ and a clock constraint c, their conjunction is defined as

$$\zeta \wedge c = \{\mathbf{v} \mid \mathbf{v} \in \zeta \wedge \mathbf{v} \models c\}$$

Forward The forward diagonal projection of a zone ζ is the zone $\nearrow \zeta$ defined as

$$\mathbf{v}' \in \nearrow \zeta \text{ iff } \exists \delta \in \mathbb{R}_{\geq 0}. \exists \mathbf{v} \in \zeta. \mathbf{v}' = \mathbf{v} + \delta \in \zeta$$

Backward The dual operator of Forward operation is the backward diagonal projection $\swarrow \zeta$:

$$\mathbf{v}' \in \swarrow \zeta \text{ iff } \exists \delta \in \mathbb{R}_{\geq 0}. \mathbf{v}' + \delta \in \zeta$$

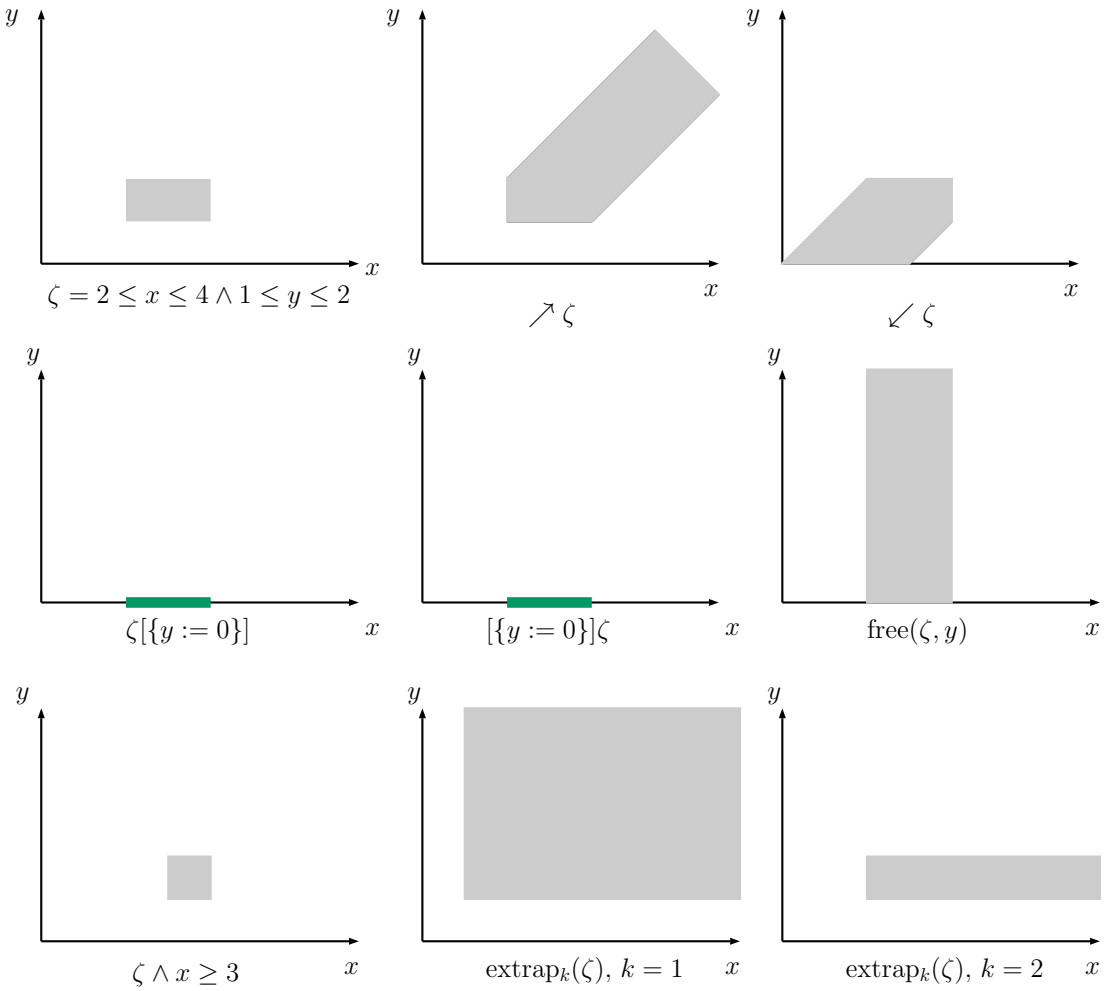


Figure 4.1: Operations on zones

Definition 4.6 (k-extrapolation)**.** *Given a constant $k \in \mathbb{N}$, a zone is said to be k-bounded if it is defined by a clock constraint C, where all clock difference bounds are smaller than k . The k-extrapolation of a zone ζ, denoted $\mathsf{extrap}_k(\zeta)$ is the smallest k-bounded zone containing ζ.*

In Figure 4.1, these operations are illustrated for a given zone $\zeta = (2 \leq x \leq 4 \wedge 1 \leq y \leq 2)$. The operation $\mathsf{free}(\zeta, y)$ performs sequentially reset operation on clock y followed by the inverse operation of reset. That is, $\mathsf{free}(\zeta, y)$ eliminates all constraints on clock y.

Many data structures have been proposed in order to represent zones and manipulate them. The Difference Bound Matrices Difference Bound Matrice (DBM) representation is the most common data structure used for timed systems analysis. We mention Clock Difference Diagrams (CDD) [LPWY99] and *Federations*

[DHLP04].

4.1.2 Difference-Bound Matrices

Difference-Bound Matrices offer a practical representation of *potential* constraints, that is to say, constraints of the form $x_i - x_j \preceq c$, where x_i and x_j are clock variables, $\preceq \in \{\leq, <\}$ and $c \in \mathbb{Z}$. The DBM representation is used to represent clock zones in the following. We consider a set of clocks $\mathcal{X} = \{x_1, x_2, \dots x_n\}$ and a constraint C over $\mathcal{X}$. Let x_0 be a reference clock variable having a constant value equal to **0**. Given a constraint C, the related zone ζ can be rewritten as a DBM , with elements m_{ij} as follows:

- For each atomic constraint $x_i - x_j \preceq c$ of ζ, let $m_{ij} = (c, \preceq)$. This includes also constraints of the form $x_i \preceq c$ and $-x_i \preceq c$ which are rewritten as $x_i - x_0 \preceq c$ and $x_0 - x_i \preceq c$, respectively.
- For each unbounded clock difference $x_i - x_j$, let $m_{ij} = \infty$, where the symbol ∞ expresses that no bound exists.
- Record the implicit constraint that every clock equals itself, that is, $x_i - x_i \leq 0$. If the correspondent matrix cell has not been already filled in by a stronger constraint, record that every clock value is non-negative, that is $x_0 - x_i \leq 0$.

If we consider as example the zone $\zeta = (x_1 \geq 10 \wedge x_2 \leq 3 \wedge x_1 - x_2 \leq 8)$, then the DBM is:

$$M = \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} (0, \leq) & (-10, \leq) & \infty \\ \infty & (0, \leq) & (8, \leq) \\ (3, \leq) & \infty & (0, \leq) \end{pmatrix} \end{array}$$

Notation 4.1 (Bounds Comparison)**.** *We introduce two operations on bounds, which are comparison and addition:*

- *(Comparison) For any constant c, $(c, \preceq) < \infty$. In addition, if $c < c'$, then $(c, \preceq) < (c', \preceq')$. Besides, $(c, <) < (c, \leq)$.*
- *(Addition) For every bound b, $b + \infty = \infty$.*
  Given two integers c_1 and c_2, $(c_1, \leq) + (c_2, \leq) = (c_1 + c_2, \leq)$ and $(c_1, <) + (c_2, \preceq) = (c_1 + c_2, <)$.

4.1.2.1 Canonical Representation of Zones

When no constraint of a zone ζ can be strengthened without reducing the number of included valuations, the zone is *closed under entailment*. For each zone ζ, there exists a unique zone ζ' such that ζ and ζ' have the same valuation set and ζ' is closed under entailment. The DBM relative to this unique zone ζ' is called the *canonical DBM representation* of ζ.

Formally, the canonical DBM representation of a zone respects this condition:

$$\forall i, j. \, m_{ij} \leq m_{ik} + m_{kj}$$

For example, the zone $\zeta = (x_1 \geq 10 \wedge x_2 \leq 3 \wedge x_1 - x_2 \leq 8)$ is not closed under entailment and its direct DBM representation violates the required condition. Precisely, $m_{20} + m_{01} = (-7, \leq) < m_{21} = \infty$. Since a zone can be interpreted by a graph, deriving the tightest constraint for a difference constraint between two clocks is equivalent to finding the shortest path between their nodes in the corresponding graph. For this purpose, Floyd-Warshall [Flo62] algorithm is used. The canonical form of the ζ zone is

$$M' = \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} (0, \leq) & (-10, \leq) & \infty \\ \infty & (0, \leq) & (8, \leq) \\ (3, \leq) & (-7, \leq) & (0, \leq) \end{pmatrix} \end{array}$$

The correspondent zone is $\zeta' = (x_1 \geq 10 \wedge x_2 \leq 3 \wedge x_1 - x_2 \leq 8 \wedge x_2 - x_1 < -7)$.
The operation of *normalization* consists in computing the canonical DBM representation of the zone.
We note that all the operations on zones introduced in Subsection 4.1.1 are implemented by use of DBM data structure.

4.2 Timed Components

Components are basically timed automata, that is finite automata extended with real-valued variables modeling the logical clocks. A finite automaton is a graph containing a finite set of locations and a finite set of edges relating them. We slightly adapt the timed automata definition from [AD94] for notation uniformity. At starting moment, the component clocks are equal to zero and they increase

synchronously at the same rate. One main feature of timed components is that the actions are timeless and instantaneous while time may pass in the locations. To restrict the behavior of a component, the actions describing transitions between two different locations are labeled with clock constraints called the guards. This means that an action is allowed only if the guard of the transition is satisfied. Besides, the locations are labeled with time progress conditions which represent a restricted form of downward closed timing constraints. A component is allowed to stay within a location as long as the time progress condition is satisfied. We note that a component has external actions, used for synchronization with other components, and internal actions which depend solely from the activity of the component independently from the others. This aspect will be further detailed in the next section.

4.2.1 Syntax

Definition 4.7 (Timed Component)**.** *A timed component is a tuple* $B = (L, A, \mathcal{X}, T, \mathsf{tpc}, s_0)$ *where*

- L *is a finite set of locations.*
- A *a finite set of actions.*
- $\mathcal{X}$ *is a finite set of local clocks.*
- $\mathsf{tpc} : L \to \mathcal{C}(\mathcal{X})$ *assigns a time progress condition to each location.*
  Time progress conditions are restricted to conjunctions of constraints as $x \leq ct$.
- $T \subseteq L \times (A \times \mathcal{C}(\mathcal{X}) \times 2^R) \times L$ *is a set of edges labeled with an action, a guard, and a set of clock reset operations in* R*, the set of all possible clock reset functions.*
- $s_0 \in L \times \mathcal{C}$ *is the initial state, i.e* $s_0 = (l_0, \bigwedge_{\mathcal{X}} x = 0)$ *where* l_0 *is the initial location. Therefore, all the clocks of the components have initially null values.*

To avoid confusion, the notation "time progress condition" (tpc) from [BS98] is preferred to "location invariant".

When $t = (l, (a, g, r), l') \in T$, the transition t can be expressed by $l \xrightarrow{a,g,r} l'$.

Intuitively, this notation defines respectively the source and destination locations l and l', the guard, the action and the set of clock reset operations happening when the transition occurs. The transition t can be executed only if its guard g is **true** for

the current valuation of clocks. An action is enabled if at least one of its transitions is enabled. Contrarily, it is disabled if all of its transitions are disabled.

Example 4.1 (A Timed Component)**.**

A timed component is depicted graphically by a box where locations are illustrated by circles. The initial location is specified by a double circle. An example of component $Controller = (L, A, \mathcal{X}, T, \mathsf{tpc}, s_0)$ is depicted in Figure 4.2 where:

- The set of locations is $L = \{lc_0, lc_1, lc_2\}$ and the initial location is lc_0.
- the set of actions is $A = \{a, c\}$
- $\mathcal{X} = \{x\}$
- $T = (t_1, t_2, t_3)$ where
  - $t_1 = (lc_0, \tau, x \geq 4, \{x := 0\}, lc_1)$
  - $t_2 = (lc_1, a, x = 4, \{x := 0\}, lc_2)$
  - $t_3 = (lc_2, c, \textbf{true}, \{x := 0\}, lc_1)$
- tpc assigns the following time progress conditions for locations: $\mathsf{tpc}(lc_0) =$ **true**, $\mathsf{tpc}(lc_1) = x \leq 4$ and $\mathsf{tpc}(lc_2) =$ **true**

We note that when a guard (respectively time progress condition) is not shown on a transition (respectively location), then it has the default value **true**. A true guard means that it suffices that the component reaches the source location for the transition to fire, without any condition on clocks. This is the case for the transition t_3 of the shown component.

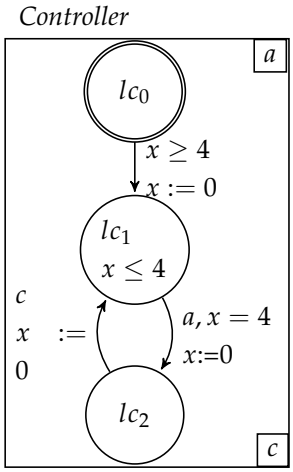


Figure 4.2: A timed component

4.2.2 Operational Semantics

For a given location and clocks valuation, the component can either execute a transition or let time elapse in the current location.

Definition 4.8 (Semantics)**.** *The semantics of a timed component* $B = (L, A, \mathcal{X}, T, \mathsf{tpc}, s_0)$ *is given by the labeled transition system* $(Q, A, \rightarrow, Q_0)$ *where* $Q \subseteq L \times \mathbf{V}$ *denotes the states of* B, $\rightarrow \subseteq Q \times (A \cup \mathbb{R}_{\geq 0}) \times Q$ *denotes the transitions according to the rules:*

- *(time progress)* $(l, \mathbf{v}) \xrightarrow{\delta} (l, \mathbf{v} + \delta)$ *if* $\left(\forall \delta' \in [0, \delta].\mathsf{tpc}(l)(\mathbf{v} + \delta')\right)$ *;*
- *(action step)* $(l, \mathbf{v}) \xrightarrow{a} (l', \mathbf{v}[r])$ *if* $\left(l, (a, g, r), l'\right) \in T$, $g(\mathbf{v}) \wedge \mathsf{tpc}(l')(\mathbf{v}[r])$.

and $Q_0 = \{(l_0, \mathbf{v}_0) | s_0 = (l_0, c_0) \wedge \mathbf{v}_0 \models c_0\}$ *denotes the initial states set.*

The time progress relation describes the passing of time, whereas the action step represents a timeless event with the possibility of clock reset. Intuitively, if the current state is $(l, \mathbf{v})$, time is allowed to elapse by δ without changing location if the time progress condition of the location l is not exceeded, i.e. $\mathsf{tpc}(l)$ evaluates to **true** for every valuation $\mathbf{v} + \delta'$, where δ' is at most δ .

Example 4.2 (Execution sequence)**.** We consider the component in Example. 4.1 where the execution sequence from the initial state $(l_0, x = 0)$ is described as follows.

- Since $\mathsf{tpc}(lc_0)$ is true, the component is allowed to stay infinitely at lc_0 location. Thus any delay transition from $(lc_0, x = 0)$ is possible, i.e. $\forall \delta \in \mathbb{R}_{\geq 0}.(lc_0, x = 0) \xrightarrow{\delta} (lc_0, x = \delta)$. However, based on the guard of t_1 , the component is allowed to leave lc_0 towards lc_1 location only if the clock x reaches 4 time units. At the execution of t, x is reset to 0.
- At location lc_1, the component Controller is not allowed to stay more than 4 units of time. When x reaches exactly 4, t_2 fires since it has the guard $x = 4$.
- Since $\mathsf{tpc}(lc_2)$ is true, the component can stay there infinitely. In addition, the transition t_3 can be instantaneous since it has a true guard and hence does not require that the clock reaches any minimum constant.

4.2.3 Symbolic Semantics

The above semantics defines a state of the system by a control location and values of the component clocks. Due to the continuous time domain, the above labeled transition system yields infinitely many states. We shall use a symbolic semantics

allowing a finite symbolic representation of the state space. We work with the zone graph as a symbolic representation. The elements of a zone graph are symbolic states. A symbolic state is a pair (l, ζ), where l is a location of the component B and ζ is a *zone*, a set of clock valuations defined by clock constraints. A symbolic state (l, ζ) represents all the states $(l, \mathbf{v})$ such that $\mathbf{v}$ satisfies ζ.
The initial configuration $s_0 = (l_0, c_0)$ corresponds to a symbolic state (l_0, ζ_0) taking into account the time progress condition of the initial location. Recursively, given a symbolic state (l, ζ), its successor with respect to a transition t of B is denoted as $\mathsf{succ}(t, (l, \zeta))$ and is defined by means of its timed and its discrete successor as follows:

- $\mathsf{time\_succ}((l, \zeta)) = (l, \nearrow \zeta \wedge \mathsf{tpc}(l))$
- $\mathsf{disc\_succ}(t, (l, \zeta)) = (l', (\zeta \wedge g)[r] \wedge \mathsf{tpc}(l'))$ if $t = (l, \_, g, r, l')$
- $\mathsf{succ}(t, (l, \zeta)) = \mathsf{extrap}_k(\mathsf{time\_succ}(\mathsf{disc\_succ}(t, (l, \zeta))))$
  where k is the maximum constant appearing in the component and for a symbolic state $s = (l, \zeta)$, $\mathsf{extrap}_k(s) = (l, \mathsf{extrap}_k(\zeta))$.

A symbolic execution of B is a sequence of symbolic states $s_0, \ldots, s_i, \ldots$. For any $i \geq 0$, there exists a transition t such that s_i reaches s_{i+1} through t. Formally, $s_i = \mathsf{succ}(t, s_{i-1})$. For a given state s, the set of states reachable from s is defined as:

$$Reach_B(s) = \{s\} \cup \bigcup_{t \in T} Reach_B(\mathsf{succ}(t, s))$$

where T is the set of transitions of the component B.

Definition 4.9 (Reachable State Set)**.** *The reachable state set $Reach_B$ of a component is the set of reachable states from its initial state: $Reach_B(\mathsf{time\_succ}(s_0))$*

In order to have a finite set of reachable state set, we apply the extrapolation operation on the zones (see Definition 4.6). It consists in making all the bounds on clocks and clock differences either bounded by a constant value or infinite. This over-approximation is sound only for timed automata without diagonal constraints [BY03, Bou04].

Example 4.3 (Infinite zone-graph and extrapolated zone-graph)**.** The timed component in Figure 4.3 has an infinite zone graph. The extrapolation with a clock ceiling k allows to have a finite zone-graph. The choice of k equal to the maximum constant 10 appearing in the component results in the finite graph depicted in Figure 4.4.

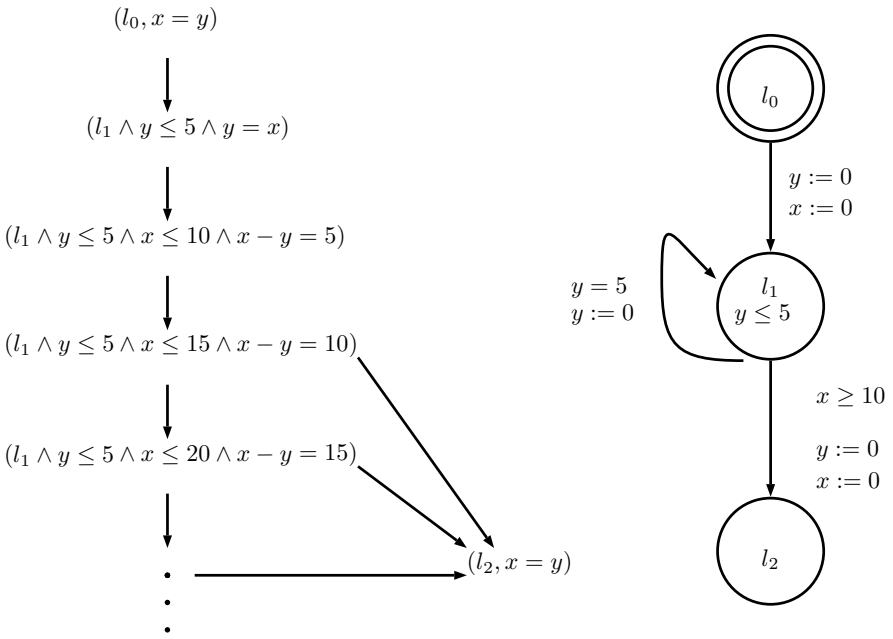


Figure 4.3: A timed component and its infinite zone-graph

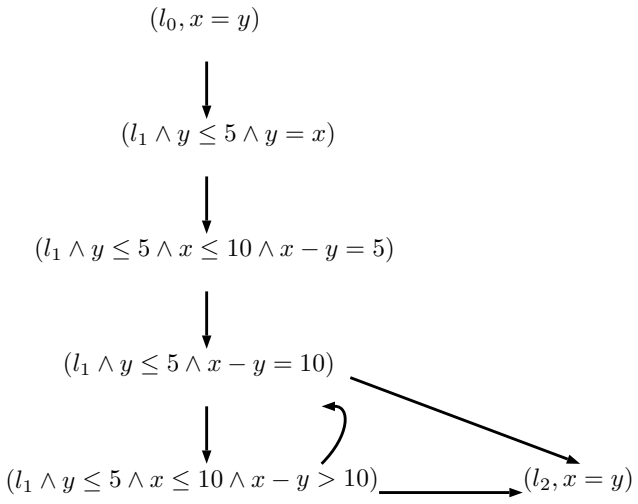


Figure 4.4: Extrapolated zone-graph for the timed component in Figure 4.3

4.3 Timed Systems

In component-based timed systems, components execute in parallel and their clocks increase synchronously. In addition, it is usually necessary to restrict the product of the behaviors of the different components in order to achieve some global properties. Components communicate by means of *interactions*, which offer synchronization between actions of different components. Systems are built from a set of components with disjoint sets of actions, locations, and clocks. Given a set of n components $\{B_i = (L_i, A_i, \mathcal{X}_i, T_i, \mathsf{tpc}_i, s_{0i})\}_{i=1...n}$, we assume that their clock sets and action sets are disjoint, i.e. for all $i \neq j$, $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$, $L_i \cap L_j = \emptyset$ and $A_i \cap A_j = \emptyset$. An interaction is defined as a subset of actions containing at most an action from each component. As in the BIP framework, an interaction is a subset

$\alpha \subseteq \cup_{i \in I} A_i$, with $I \subseteq \{1 \dots n\}$ such that $\forall i \in I$, $|\alpha \cap A_i| \leq 1$

We denote by $involved(\alpha)$, the set of components which have one action participating in interaction α. Formally,

$$involved(\alpha) = \{B_i | A_i \cap \alpha \neq \emptyset\}$$

Interactions and their enabledness conditions are defined as in the BIP framework.

Definition 4.10 (Timed System)**.** *Given n components $B_i = (L_i, A_i, \mathcal{X}_i, T_i, \mathsf{tpc}_i, s_{0i})$ and a set of interactions γ, we define the timed system $\|_\gamma B_i$ as the component $(L, \gamma, \mathcal{X}, T_\gamma, \mathsf{tpc}, s_0)$, where*

- $L = L_1 \times L_1 \times \cdots \times L_n$ *is the set of global locations.*
- $s_0 = ((l_{01}, ..., l_{0n}), \bigwedge_i c_{0i})$ *, where* $\forall i$, $s_{0i} = (l_{0i}, c_{0i})$.
- $\mathcal{X} = \cup_i \mathcal{X}_i$ *is the set of clocks.*
- $\mathsf{tpc}(\bar{l}) = \bigwedge_i \mathsf{tpc}(l_i)$, $\forall \bar{l} = (l_1, ..., l_n) \in L$.
- *The set of global transitions is defined by*

$$T_\gamma = \left\{ (\bar{l}, (\alpha, g, r), \bar{l}') \;\middle|\; \begin{array}{l} \bar{l} = (l_1, ..., l_n) \in L,\ \ \bar{l}' = (l'_1, ..., l'_n) \in L \\ \alpha = \{a_i\}_{i \in I} \in \gamma,\ \ \forall i \in I. (l_i, (a_i, g_i, r_i), l'_i) \in T_i,\ \ \forall i \notin I. l_i = l'_i \\ \qquad r = \bigcup_{i \in I} r_i, g = \bigwedge_{i \in I} g_i \end{array} \right\}$$

The obtained behavior $\|_\gamma B_i$ can execute an interaction $\alpha \in \gamma$ if for each $B_i \in involved(\alpha)$, the action $A_i \cap \alpha$ is enabled in B_i. Therefore, in $\|_\gamma B_i$, a component B_i can execute an action a_i only as part of an interaction to which it belongs, along with the execution of all the other actions belonging to the same interaction. This requires that all the guards of the involved transitions are satisfied. As

for internal actions, i.e. actions forming unary interactions, the transitions are executed without association with any action belonging to another component.

Example 4.4 (A Timed System)**.** Figure 4.5 illustrates a timed system composed of a *Controller* component interacting with two *Worker* components. The three components are synchronized through interaction set $\gamma = \{a|b_0, a|b_1, c|d_0, c|d_1\}$. The initial global location of the system is (lc_0, l_1^0, l_1^1). When x is at least 8, the *Controller* component takes an internal transition and resets x to reach the lc_1 location. When the Controller reaches 4 time units at lc_1, the transition labeled with a should fire. Since action a is shared between the two interactions $a|b_0$ and $a|b_1$, either b_0 or b_1 should be executed. This requires that at least one of the clocks y_0 and y_1 should reach 8 units of time when x is equal to 4. When the Controller is at lc_2 location, the execution of c action requires the parallel execution of b_0 or b_1. Therefore, it suffices that one of the worker components is at its l_2 location (l_2^0 or l_2^1), since c, d_0 and d_1 have no guards on transitions.

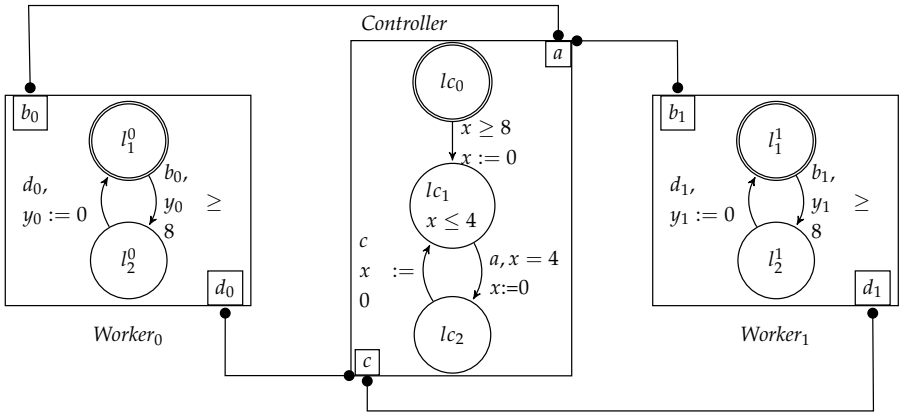


Figure 4.5: A timed Controller-Workers system

Remark 4.1 (Semantics of a Timed System)**.** *The semantics of the system $\|_\gamma B_i$ resulting from the composition of the components $B_i = (L_i, A_i, \mathcal{X}_i, T_i, \mathsf{tpc}_i, s_{0i})$ corresponds to the semantics of the component $(L, \gamma, \mathcal{X}, T_\gamma, \mathsf{tpc}, s_0)$ defined in Definition 4.10. It is therefore defined as the labeled transition system $(Q, \gamma, \rightarrow_\gamma, Q_0)$, where $Q \subseteq L \times \mathbf{V}$ denotes the states of $\|_\gamma B_i$, Q_0 is the set of global initial states and $\rightarrow_\gamma \subseteq Q \times (\gamma \cup \mathbb{R}_{\geq 0}) \times Q$ denotes the transitions according to the following rules*

- *(time progress)*

$$\frac{\delta \in \mathbb{R}_{\geq 0}, \ \ \forall i \in \{1 \ldots n\}. \, \mathsf{tpc}(l_i)(\mathbf{v} + \delta)}{(l, \mathbf{v}) \xrightarrow{\delta} (l, \mathbf{v} + \delta)}$$

- *(interaction execution)*

$$\frac{\alpha = \{a_i\}_{i\in I},\ \forall i \in I.\,(l,\mathbf{v}) \xrightarrow{a_i} (l',\mathbf{v}[r_i]),\ \forall i \notin I.\,l = l',\ g(\mathbf{v}) \wedge \mathsf{tpc}(\bar{l}')(\mathbf{v}[r])}{(\bar{l},\mathbf{v}) \xrightarrow{\alpha} (\bar{l}',\mathbf{v}[r])}$$

As for the notations $\bar{l}$, $\bar{l}'$, g and $\mathsf{tpc}(\bar{l})$, we follow those in definition.4.10 where they are defined, respectively, as the global source location of the system, the destination location, the global guard, and the time progress condition corresponding to the global location $\bar{l}$.

4.4 Timed Properties

4.4.1 Safety Properties and Invariants

We are interested in verifying invariant safety properties, that is properties describing that some 'bad state should never be reached'. For example, for safety reasons, in a railway system, a train should never enter while the gate is not down. We note the existence of safety properties that are not invariants. For instance, 'component B_1 can execute action a only after component B_2 executes action b' is not an invariant since it is not a state property.

4.4.1.1 Invariants

An invariant is a predicate ϕ that holds for every reachable state of the system. The predicate ϕ should be fulfilled in the initial state and its satisfaction should be preserved along the execution of transitions. This means that if ϕ holds at a source state of a transition, then it holds also for the destination state.

Definition 4.11 (Invariant of a Component)**.** *Given a component $B = (L, A, \mathcal{X}, T, \mathsf{tpc}, s_0)$, a state predicate ϕ is an invariant of B, denoted $inv(B,\phi)$ if it is satisfied by every reachable state of the system.*
Formally, ϕ is an invariant if it is satisfied by the initial state and for any reachable state s and transition $t \in T$:

$$(s \models \phi \wedge s \xrightarrow{t} s') \Rightarrow s' \models \phi$$

Proposition 4.1. *Given two invariants ϕ_1 and ϕ_2 of a component B, then $\phi_1 \vee \phi_2$ and $\phi_1 \wedge \phi_2$ are also invariants of B.*

It is possible to prove that a predicate ϕ is an invariant of a given component B by finding a stronger predicate, that is a predicate implying it, which is an invariant of the component. This is stated formally in the following proposition

$$((\phi_{strong} \Rightarrow \phi) \wedge inv(B, \phi_{strong})) \Rightarrow inv(B, \phi)$$

4.4.1.2 Safety Properties

State safety properties express that combinations of locations and clock constraints are reachable or unreachable from the initial state. Formally, the checked properties Ψ are represented by logical formulas combining locations of the components and clock constraints following this grammar

$$\Psi ::\equiv a \mid at(l) \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi$$

where a is an atomic clock constraint and $at(l_i)$ is a predicate expressing the presence of component B_i at location l_i. We note that the predicate $\Psi_1 \vee \Psi_2$ (resp. $\Psi_1 \Rightarrow \Psi_2$) is included in this grammar since it is equivalent to $\neg(\neg\Psi_1 \wedge \neg\Psi_2)$ (resp.$\neg\Psi_1 \vee \Psi_2$). Intuitively, a property Ψ is satisfied if all the reachable states of the system satisfy it.

4.4.2 Deadlock-freedom

Deadlock-freedom is a significant property for concurrent systems safety. It indicates their ability to always perform some action. The presence of deadlock is one common cause of errors that hinders systems consisting of concurrent components, that is components sharing common resources or which follow strongly constraining synchronization. In fact, strong synchronization serves to constraint the global behavior of the system and to eliminate undesirable global locations, but this may cause the appearance of deadlock states. Furthermore, in the case of real-time systems, time brings another degree of concurrency. In fact, the presence of time progress conditions which should not be exceeded at components locations and the necessity of a guard achievement for a transition to occur may provoke conflicting requirements over two components or more, making the execution of some global actions impossible. A deadlocked state of a system implies that neither an unary interaction nor a global transition triggered

by a multi-party interaction synchronizing many components are enabled. The deadlocks therefore depend on the enabledness of the interactions. Accordingly, we first introduce the enabledness condition of an interaction.

Interation Enabledness

For an interaction α of the system, $\alpha \in \gamma$, the predicate characterizing the enabledness of α is defined as

$$enabled(\alpha) \equiv \bigvee_{T \in trans(\alpha)} enabled(T)$$

where the set $trans(\alpha)$ contains the transitions of the system which are triggered by α interaction. That is, $enabled(\alpha)$ expresses the states from which interaction α can be executed. It is the case when all of its actions are ready for synchronizing and when for each one of those actions, one transition is enabled.

Example 4.5 (Enabledness predicate for an interaction)**.** If we consider the interaction $\alpha = a|b_0$ of the system depicted in Figure 4.5, then the enabldness predicate $enabled(\alpha)$ is expressed as

$$enabled(\alpha) \equiv enabled(T) \vee enabled(T')$$

where $(lc_1, l_1^0, l_1^1) \xrightarrow{T} (lc_2, l_2^0, l_1^1)$ and $(lc_1, l_1^0, l_2^1) \xrightarrow{T'} (lc_2, l_2^0, l_2^1)$

4.4.2.1 Global Transition Enabledness

Let us consider a global transition $T \in T_\gamma$, $T = (\bar{l}, (\alpha, g, r), \bar{l}')$, where $\bar{l} = (l_1, ..., l_n)$ is the global source location, $\bar{l}' = (l_1', ..., l_n')$ is the global destination location and $\alpha = \{a_i\}_{i \in I}$. The guard and the tpc of the global location follow Definition.4.10. The predicate characterizing the enabledness of T is

$$enabled(T) \equiv at(\bar{l}) \wedge \mathsf{tpc}(\bar{l}) \wedge \swarrow (g \wedge \mathsf{tpc}(\bar{l}')[r])$$

This predicate implies that for a transition to be enabled, not only the time progress conditions of the source and destination locations of the components involved in the transition are taken into account, but also the time progress conditions of the non involved components. In other words, due to the synchronous nature of time, the enabledness of a transition is not projected to the involved components

only. The clocks of the other components do also increase with the same time elapse and the time progress conditions of their current locations should not be exceeded when the guard and tpc of the involved components would enable the global transition.

Example 4.6 (Enabledness predicate for a transition)**.** We consider the example in Figure 4.5 and the transition $T = (\bar{l}, (\alpha, g, r), \bar{l}')$, where $\bar{l} = (lc_1, l_1^0, l_1^1)$, $\bar{l}' = (lc_2, l_2^0, l_1^1)$, $\alpha = a|b_0$, $g = (y_0 \geq 8 \wedge x = 4)$ and $r = \{x := 0\}$. The time progress conditions of the two global locations are $\mathsf{tpc}(\bar{l}) = x \leq 4$ and $\mathsf{tpc}(\bar{l}') = \mathbf{true}$. The enabledness predicate of T is

$$enabled(T) \equiv at(\bar{l}) \wedge x \leq 4 \wedge \swarrow (y_0 \geq 8 \wedge x = 4) \equiv at(\bar{l}) \wedge x \leq 4 \wedge y_0 - x \geq 4$$

For a more general case, we consider the system in Figure 4.6 which differs from Figure 4.5 in that the location l_1^1 of component $Worker_1$ has a tpc equal to $x \leq 10$. We consider the transition $T = (\bar{l}, (\alpha, g, r), \bar{l}')$, where $\bar{l} = (lc_1, l_1^0, l_1^1)$, $\bar{l}' = (lc_2, l_2^0, l_1^1)$, $\alpha = a|b_0$, $g = (y_0 \geq 8 \wedge x = 4)$ and $r = \{x := 0\}$. The component $Worker_1$ does not participate in transition T. However, $\mathsf{tpc}(l_1^1)$ is taken into account for the computation of the enabledness predicate for the global transition T, since (l_1^1) belongs the global destination location :

$$\begin{aligned} enabled(T) &\equiv at(\bar{l}) \wedge x \leq 4 \wedge y_1 \leq 10 \wedge \swarrow (x = 4 \wedge y_0 \geq 8 \wedge y_1 \leq 10) \\ &\equiv at(\bar{l}) \wedge x \leq 4 \wedge y_1 \leq 10 \wedge y_0 - x \geq 4 \wedge y_1 - x \leq 6 \end{aligned}$$

Definition 4.12 (Deadlock States Set)**.** *The predicate DIS characterizing the set of all the states from which all the interactions are disabled is defined as*

$$DIS \equiv \bigwedge_{\alpha \in \gamma} \neg enabled(\alpha)$$

Example 4.7 (Predicate Characterizing Deadlocked States)**.** For the system in Figure 4.5, the deadlock states predicate is

$$DIS \equiv \neg enabled(a|b_0) \wedge \neg enabled(a|b_1) \wedge \neg enabled(c|d_0) \wedge \neg enabled(c|d_1)$$

4.5 Systems with Data

For better expressivity and applicability of the system model, the timed components introduced in Definition 4.7 are extended with data variables. Similarly to

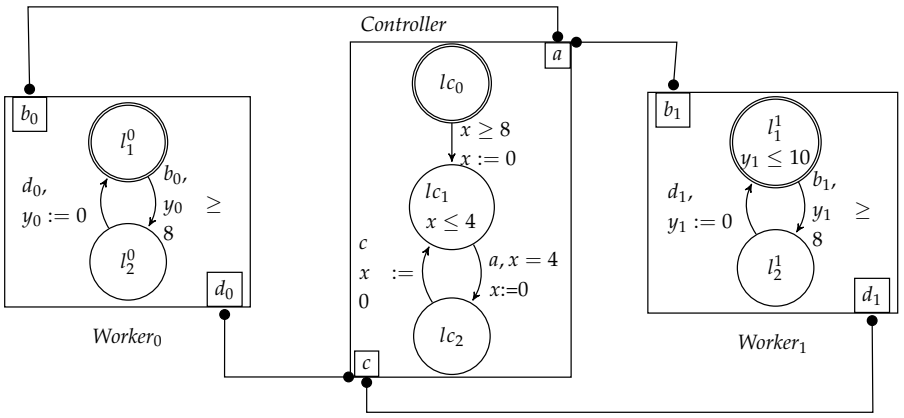


Figure 4.6: A timed Controller-Workers system with a tpc at l_1^1 location

clock variables, data variables may appear in the guard of transitions as enabling conditions. In addition, their value may be updated when the transition fires.

Definition 4.13 (Guards on clocks and data variables)**.** *Let $\mathcal{X}$ be a set of clock variables and $\mathcal{D}$ a set of data variables. We use $\mathcal{G}(\mathcal{X}, \mathcal{D})$ to denote the set of guards g, generated by the following grammar:*

$$g ::= g_c | g_d | g \wedge g$$

where g_c is an atomic constraint of the form $x\#ct$, where $x \in \mathcal{X}$, $\# \in \{<, \leq, =, \geq, >\}$ and $ct \in \mathbb{N}$, and where g_d is a predicate on a set of data variables belonging to $\mathcal{D}$.

Thereby, a guard can be perceived as a conjunction of two families of guards: guards on the clocks and guards on the data variables.

We enlarge the notion of valuation to take into account the data variables. Valuations assign values for data variables in addition to the clocks. The satisfaction relation $\mathbf{v} \models C$ extends in the straightforward way to data variables. As previously mentioned, clock valuations increase simultaneously at the same rate. However, data variables are insensitive to the passing of time. Formally,

- $(\mathbf{v} + \delta)(x) = (\mathbf{v})(x) + \delta$ for every clock $x \in \mathcal{X}$
- $(\mathbf{v} + \delta)(d) = (\mathbf{v})(d)$ for every variable $d \in \mathcal{D}$

Assignment operations are defined for clocks and data variables. We denote by R the set of all reset operations over elements of $\mathcal{X}$ and update operations over elements of $\mathcal{D}$.

Definition 4.14 (Extended Timed Component)**.** *An extended timed component is a tuple* $B_e = (L, A, \mathcal{X}, \mathcal{D}, T, \mathsf{tpc}, s_0)$ *where*

- L, A, $\mathcal{X}$, tpc *and* s_0 *are defined as in Definition 4.7, that is:*
  - L *is a finite set of locations.*
  - A *a finite set of actions.*
  - $\mathcal{X}$ *is a finite set of local clocks.*
  - $\mathsf{tpc} : L \to \mathcal{C}$ *assigns a time progress condition to each location, where* C *is a set of downwards close timing constraints, in the form* $x \leq c$ *or* $x < c$*.*
  - $s_0 \in L \times \mathcal{C}$ *is the initial state.*
- $\mathcal{D}$ *is a finite set of data variables.*
- $T \subseteq L \times (A \times \mathcal{G}(\mathcal{X}, \mathcal{D}) \times 2^R) \times L$ *is a set of edges labeled with an action, a guard, and a set of reset and update operations over clocks and data variables in* $\mathcal{D}$*.*

Referring to Definition 4.7, the main extension consists in introducing the data variable set $\mathcal{D}$, in increasing the restrictiveness of guards by putting enabling condition on data and in updating data when transitions fire. Concretely, a transition $t = (l, (a, g, r), l') \in T$ can be executed only if its guard g evaluates to true for the current valuation of clocks and data variables. The set $r \subseteq R$ contains respectively reset and update operations on clocks and data variables. These operations are performed when the transition t fires.

Example 4.8 (An Extended Timed Component)**.** In Figure 4.7, an extended timed component with actions a, b and c is depicted. The data set is $\mathcal{D} = \{\mathrm{k}\}$, the guard on the transition labeled with a is $(x = 4 \wedge \mathrm{k} \leq 5)$. The update operations on a is $r = \{x := 0, \mathrm{k}{:=}\mathrm{k}{+}20\}$, whereas the update operation on the transition b is $\mathrm{k}{:=}\mathrm{k}{-}3$.

Extended timed components can be composed together through interactions in order to form a composite extended timed system.

Definition 4.15 (Extended Timed System)**.** *Given n extended components* $B_{e,i} = (L_i, A_i, \mathcal{X}_i, \mathcal{D}_i, T_i, \mathsf{tpc}_i, s_{0i})$ *and an interaction set* γ*, we define the extended timed system* $\|_\gamma B_{e,i}$ *as the timed component* $(L, \gamma, \mathcal{X}, \mathcal{D}, T_\gamma, \mathsf{tpc}, s_0)$*, where*

- L, s_0, $\mathcal{X}$ *and* tpc *are defined as in Definition 4.10.*

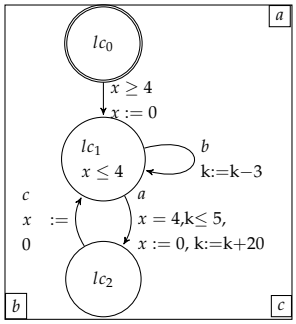


Figure 4.7: An extended timed component

- *The set of data variables is* $\mathcal{D} = \cup_i \mathcal{D}_i$
- *The global transitions are extended in a straightforward way to timed systems by considering the particularity that the guards put enabling conditions on data variables in* $\mathcal{D}$ *and that update operations on transitions affect them.*

The semantics of an extended timed system extends the semantics introduced in Definition 4.10 in that an interaction takes place if the guards on data variables are also valid and in that their values are affected following the update operations on transitions. We note that the data transfer on interactions is defined as in the BIP framework.

4.6 Compositional Verification: Naive Approach

In Chapter 3, a compositional verification method was proposed for untimed systems modeled in BIP language. It is based on the computation of an invariant over-approximating the reachable state set and the application of the deductive approach. The global invariant is computed as the conjunction of local invariants of the different components and an interaction invariant deduced from the interaction structure which ensures their synchronization. The component invariant $CI(B_i)$ is computed as the reachable state set of the component. In the following, we propose an adaptation of the rule to timed systems. The computation of interaction invariant may be automatically adapted for timed systems by abstracting it totally from the timing characteristics of the components and applying the already introduced methods for the computation of $II(\gamma)$ in case of untimed systems. A direct application of this rule to timed systems consists in redefining the computation of the component invariant. In Section 4.2 of this chapter, we introduced a method

for the computation of the reachable state set $Reach_B$ of a given component B.

and how to incorporate their effect in the global invariant in a way that accumulates with the following method for handling with time.
In order to illustrate the global invariant computation in the timed case, we consider the system in Figure 4.8. It is constituted of a controller synchronized with one worker through interactions $a|b_1$ and $c|d_1$.

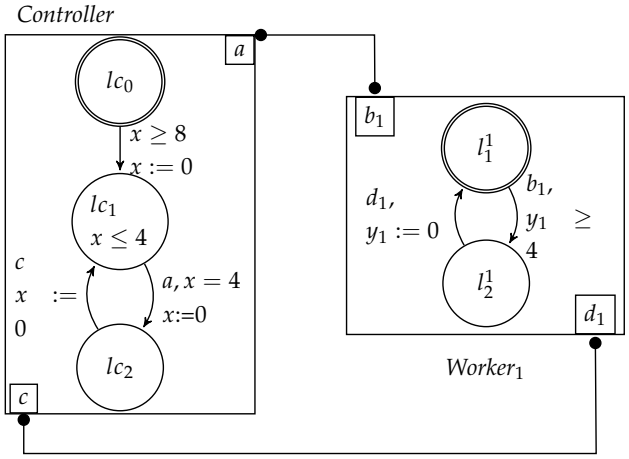


Figure 4.8: A timed system

The computation of the reachable state set for the two components results in the following component invariants.

$$CI(Controller) \equiv lc_0 \vee (lc_1 \wedge x \leq 4) \vee lc_2$$

$$CI(Worker_1) \equiv l_1^1 \vee (l_2^1 \wedge y_1 \geq 4)$$

The interaction invariant for this system is

$$II \equiv (lc_0 \vee lc_1 \vee lc_2) \wedge (l_1^1 \vee l_2^1) \wedge (lc_0 \vee lc_1 \vee l_2^1) \wedge (lc_2 \vee l_1^1)$$

Now we want to check the following property: $\Psi \equiv lc_1 \wedge l_1^1 \Rightarrow y_1 - x \geq 0$. In fact, the property is valid because when the interaction $c|d_1$ occurs, both clock x and clock y_1 are reset at the same time. Also, when the controller component first reaches lc_1 location from lc_0 location, the clock x is reset whereas the clock y_1 in the worker component have not been reset since the start time. The global invariant is:

$$GI \equiv (lc_0 \wedge l_1^1) \vee (lc_1 \wedge x \leq 4 \wedge l_1^1) \vee (lc_2 \wedge l_2^1 \wedge y_1 \geq 4)$$

Unfortunately, *GI* is not strong enough to prove the property Ψ. In fact, the clocks of the two components increase with the same rate. This engenders stronger synchronization which neither of the above invariants can capture. In some way, with the current version of the global invariant, the method deals with the components as if their clocks are totally independent while the time model is synchronous. For instance, two clocks which are reset at the same time remain equal until one of them is updated.
This example highlights the limitation of the above invariant and reveals the need to time synchronization capturing invariants.

4.7 Summary

In this chapter, we presented a component-based framework for modeling real-time systems. A timed system is conceived as a set of components equipped with local clocks and glued together through a set of multi-party interactions. The application of the invariant computation method proposed with the D-Finder tool reveals the need for timed invariants allowing to catch properties that are induced by the synchronous model of time. In fact, the time-synchronization aspect constrains further the system behavior but its effect would not be captured by neither the components invariants which reflect the independent behavior of each component, nor by the interaction invariant ignoring totally the synchronous model of time.
In the next chapter, we introduce additional types of invariants tailored to timed systems at the precise goal of capturing the time-synchronization between the different components.

Chapter 5

Compositional Invariant Generation

Contents

By the end of the previous chapter, we illustrated the main limitation of the compositional verification method which was previously proposed for untimed systems. The difficulty is that the global invariant does not consider the synchronous model of time and therefore does not reflect the effects of the time synchronization between the clocks of the different components. This limitation is a major hurdle for the application of compositional methods in the framework of real-time systems. In the above method, the interaction invariant abstracts totally away the information about the clocks while the component invariant reflects its independent behavior in isolation from the other components. Ideally, a sufficiently strong global invariant for timed systems should track the instants where actions from

the different components are synchronized, precisely through the execution of interactions. These dates of synchronization should be also reflected in the local invariants of the different components.
In order to capture this dual information, we propose to add auxiliary clocks, called *history clocks*, as follows. To each action a, we add an action history clock h_a which is reset whenever action a occurs. On the one hand, since an interaction synchronizes actions from different components, their history clocks are reset at the same time and their values remain equal as they increase at the same rate, unless an action related to one of them occurs again during the execution of a more recent interaction. As a result, relations between the history clocks of the different components are deduced from the interaction structure and gathered in a new type of invariant. On the other hand, the history clock of an action a and the related reset operations are taken into account during the computation of the invariant of the component containing a. Consequently, the local invariant of each component extended with history clocks contains relations between the added history clocks and the inner local clocks. By transitivity, relations between the inner clocks of the different components are induced, serving to strengthen the over-approximation of the reachable states set that is expressed by the global invariant.
Besides the history clocks for actions, we introduce *history clocks for interactions* which record the last occurrences of interactions. New invariants relating them to each other and to the history clocks for actions are proposed and they generally offer a stronger global invariant in case of conflicting interactions.
In this chapter, we formalize the notion of history clocks and detail the computation of the new invariants.

5.1 History Clocks for Actions

In this section, we extend the components with the auxiliary history clocks at the hope that the verification rule yields more successful application on timed systems. For each action, we relate an action history clock which is reset whenever the action occurs. When an interaction is executed, the history clocks of all the participating actions are reset simultaneously. Therefore, a new invariant gathering relations between the different history clocks can be generated based on the interactions structure. Together with the invariants of the extended components and the interaction invariant, it forms generally a stronger global invariant.
Definition 5.1 (Components with History Clocks)**.**

Given a component $B = (L, A, \mathcal{X}, T, \mathsf{tpc}, s_0)$*, its extension with history clocks is the component* $B^h = (L, A, \mathcal{X} \cup \mathcal{H}_A, T^h, \mathsf{tpc}, s_0^h)$ *where*

- $\mathcal{H}_A = \{h_0\} \cup \{h_a \mid a \in A\}$ *is the set of history clocks,*
- $T^h = \{(l, (a, g, r \cup \{h_a\}), l') \mid (l, (a, g, r), l') \in T\}$,
- $s_0^h = (l_0, c_0^h)$*, where* $c_0^h = (c_0 \wedge h_0 = 0 \wedge \bigwedge_{a \in A} h_a > 0)$*, given* $s_0 = (l_0, c_0)$.

For simplicity and in order to focus on the issues relevant to timed systems, we abstract away the data variables from the verification method in the following of this dissertation.

5.1.1 Invariants for Action History Clocks

The history clocks are added for verification purpose and do not influence the behavior of the system, since there is no guard and no progress condition defined on their values. They appear exclusively in the reset operations. We deduce that the component extended with history clocks is bisimilar to the original component. Besides, the invariant of the extended component is also an invariant of the original one. Abusing notation, for a set $A = \{a_1, ..., a_m\}$ of actions, we denote by $\exists \mathcal{H}_A$ the notation $\exists h_{a_1} \exists h_{a_2} \ldots \exists h_{a_m} \exists h_0$.

Proposition 5.1. *If* Φ^h *is an invariant of the component* B^h *extended from* B *with history clocks, then* $\Phi = \exists \mathcal{H}_A.\Phi^h$ *is an invariant of* B.

Proof.
We decompose the proof in two steps in order to show that $\Phi = \exists \mathcal{H}_A.\Phi^h$ is an overapproximation of $Reach_B(s_0)$.

1. Any symbolic state (l, ζ^h) in the set $Reach_{B^h}(s_0^h)$ of the reachable states of B^h corresponds to a symbolic state (l, ζ) in $Reach_B(s_0)$. It can be obtained by the projection of ζ^h on the set of original clocks $\mathcal{X}$ where the location remains unchanged. Formally, ζ is equal to $\exists \mathcal{H}_A.\zeta^h$.
   It results that $\exists \mathcal{H}_A.Reach_{B^h}(s_0^h) \equiv Reach_B(s_0)$.
2. Any invariant ϕ^h of the component B^h is an over-approximation of $\exists \mathcal{H}_A.Reach_{B^h}(s_0^h)$:

$$\exists \mathcal{H}_A.Reach_{B^h}(s_0^h) \subseteq \exists \mathcal{H}_A.\Phi^h$$

The conjunction of (1) and (2) proves that $\Phi = \exists \mathcal{H}_A.\Phi^h$ is an invariant of B.

When an interaction is executed, all the participating actions are reset. It results that their values are smaller than the history clocks of the non executed actions.

Besides, they remain equal until a more recent interaction containing one of them is executed. The relation that relates history clocks of the different components participating in the different interactions bring in an additional invariant which is defined in the following.

Definition 5.2 (History Clocks Inequalities)**.** *Given an interaction set γ containing all the interactions of a timed system resulting from the parallel composition of a set of components, the history clocks inequalities predicate $\mathcal{E}(\gamma)$ is defined as follows:*

$$\mathcal{E}(\gamma) \equiv \bigvee_{\alpha\in\gamma}\Big(\big(\bigwedge_{\substack{a_i,a_j\in\alpha\\ a_k\in Act(\gamma\ominus\alpha)}} h_{a_i} = h_{a_j} \leq h_{a_k}\big) \wedge \mathcal{E}(\gamma\ominus\alpha)\Big).$$

*where $\gamma\ominus\alpha = \{\beta\setminus\alpha \mid \beta\in\gamma \wedge \beta \not\subseteq \alpha\}$ and $\mathcal{E}(\emptyset) =$ **true**.*

The predicate $\mathcal{E}(\gamma)$ characterizes relations between the history clocks at different execution scenarios and can be understood as follows. If we suppose that $\alpha\in\gamma$ is the most recent interaction of the system, then lately all the history clocks of its participating actions are reset at the same time. In addition, they are smaller than all of the other history clocks, contained in $\gamma\ominus\alpha$, since α is the most recent interaction. This predicate is recursively expressed for the remaining set of interactions $\gamma\ominus\alpha$ which eliminates from any interaction β in γ all the actions from α. As an illustration, for $\beta = (a \mid a_1 \mid a_2)$, $\alpha = (a_1 \mid a_2)$ and $\gamma = \{\alpha,\beta\}$, $\gamma\ominus\alpha = \{a\}$.

The function "min" can be used as syntactic sugar to have a more compact formulation for $\mathcal{E}(\gamma)$:

$$\mathcal{E}(\gamma) \equiv \bigvee_{\alpha\in\gamma}\Big(\bigwedge_{a_i,a_j\in\alpha} h_{a_i} = h_{a_j} \leq \min_{a_k\in Act(\gamma\ominus\alpha)} h_{a_k} \wedge \mathcal{E}(\gamma\ominus\alpha)\Big).$$

As an example, for $\gamma = \{(a \mid b_1),(c \mid d_1)\}$ modeling the interactions between the controller and the worker from Figure 4.8, a more compact form is obtained:

$$\mathcal{E}(\gamma) \equiv \big(h_a = h_{b_1} \leq \min(h_c, h_{d_1}) \wedge h_c = h_{d_1}\big) \vee \big(h_c = h_{d_1} \leq \min(h_a, h_{b_1}) \wedge h_a = h_{b_1}\big).$$

This predicate constraining the history clocks inequalities is inductive for $\|_\gamma B_i^h$.

Proposition 5.2. *$\mathcal{E}(\gamma)$ is an inductive assertion of $\|_\gamma B_i^h$.*

Proof.
We reason by induction on the execution sequence. We assume that $\mathcal{E}(\gamma)$ is satisfied at an arbitrary state s of the system and proceed to prove that it is satisfied by

all of its successor states. There are two types of successors: time successors and discrete successors. As for time successors, all the relations between the clocks are preserved as they do all increase at the same rate. We propose to prove that $\mathcal{E}(\gamma)$ is satisfied for any discrete successor of s.
We have the following equivalence by definition:

$$\mathcal{E}(\gamma \ominus \alpha) \equiv \bigvee_{\alpha' \in \gamma \ominus \alpha} \Big(\bigwedge_{a_i, a_j \in \alpha'} h_{a_i} = h_{a_j} \leq \min_{a_k \in Act(\gamma \ominus \alpha')} h_{a_k} \wedge \mathcal{E}(\gamma \ominus \alpha') \Big)$$
$$\equiv \exists \alpha' \in \gamma. \bigwedge_{a_i, a_j \in \alpha' \wedge a_i, a_j \notin \alpha} h_{a_i} = h_{a_j} \leq \min_{a_k \in Act(\gamma \ominus (\alpha \cup \alpha'))} h_{a_k} \wedge \mathcal{E}(\gamma \ominus (\alpha \cup \alpha'))$$

We suppose that $\mathcal{E}(\gamma)$ holds at state s, then the following predicate is satisfied at s:

$$\exists \alpha' \in \gamma. \Big(\bigwedge_{a_i, a_j \in \alpha'} h_{a_i} = h_{a_j} \leq \min_{a_k \in Act(\gamma \ominus \alpha')} h_{a_k} \wedge \mathcal{E}(\gamma \ominus \alpha') \Big) \quad (Eq.1)$$

If the interaction α, $s \xrightarrow{\alpha} s'$, occurs then all the history clocks related to its participating actions are reset while the other clocks are not modified:

$$\bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j} = 0 \leq \min_{a_k \in Act(\gamma \ominus \alpha)} h_{a_k} \quad (Eq.2)$$

When the state s' is reached through interaction α, all the clocks outside α are preserved and they are bigger than those which are reset. Therefore, the formula $(Eq.1)$ which is satisfied at s state is updated when executing α, and its conjunction with $(Eq.2)$ infers the following predicate which is satisfied at state s':

$$\underbrace{\begin{array}{l} \bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j} = 0 \leq \min_{a_k \in Act(\gamma \ominus \alpha)} h_{a_k} \wedge \\ \exists \alpha' \in \gamma. \bigwedge_{a_i, a_j \in \alpha' \wedge a_i, a_j \notin \alpha} h_{a_i} = h_{a_j} \leq \min_{a_k \in Act(\gamma \ominus (\alpha \cup \alpha'))} h_{a_k} \wedge \mathcal{E}(\gamma \ominus (\alpha \cup \alpha') \end{array}}_{(3)}$$

The underlined sub-formula is nothing but the definition of $\mathcal{E}(\gamma \ominus \alpha)$ as shown at the beginning of this proof. This implies that formula (3) is equivalent to the following predicate, which is by consequence satisfied at s' state.

$$\bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j} = 0 \leq \min_{a_k \in Act(\gamma \ominus \alpha)} h_{a_k} \wedge \mathcal{E}(\gamma \ominus \alpha)$$

We note that this predicate is valid immediately after the execution of α interaction. It is a subformula of $\mathcal{E}(\gamma)$ where the clocks of α are still equal to zero. The predicate $\mathcal{E}(\gamma)$ is therefore preserved by discrete transitions.

By increasing all the history clocks in the above formula by the same rate, $\mathcal{E}(\gamma)$ remains valid. We deduce that $\mathcal{E}(\gamma)$ holds at s' state and its time successors.

5.1.2 Strengthening the Global Invariant

In order to obtain a stronger invariant, we propose to conjunct $\bigwedge_i CI(B_i^h) \wedge II(\gamma)$ with $\mathcal{E}(\gamma)$. The obtained predicate is also an invariant.

Proposition 5.3. *If Φ_1^h is an invariant of B^h and Φ_2^h is an inductive assertion on B^h expressed on history clocks $\mathcal{H}_A$, then $\Phi = \exists \mathcal{H}_A.(\Phi_1^h \wedge \Phi_2^h)$ is an invariant of B.*

Proof. We first show that Φ_2^h is an invariant of a component B_ϕ^h which is an extension of B with history clocks in $\mathcal{H}_A$ and where the initial state satisfies the predicate Φ_2^h. More precisely, if the initial state of B is (l_0, c_0), the initial state of B_ϕ^h is defined as $(l_0, c_0^h \wedge \Phi_2^h)$. It can be shown that $(\Phi_1^h \wedge \Phi_2^h)$ is an invariant of B_ϕ^h. Following the same reasoning in the proof of Proposition 5.1, it is proven that $\Phi = \exists \mathcal{H}_A.(\Phi_1^h \wedge \Phi_2^h)$ is an invariant of B.
By taking $\bigwedge_i CI(B_i^h) \wedge II(\gamma)$ for Φ_1^h and $\mathcal{E}(\gamma)$ for Φ_2^h, this proposition implies the following corollary.

Corollary 5.1. *$\Phi \equiv \exists \mathcal{H}_A.(\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma))$ is an invariant of $\|_\gamma B_i$.*

In general, this invariant is tighter than $\bigwedge_i CI(B_i) \wedge II(\gamma)$ and offers more successful applications of the verification rule.

Example 5.1. We reconsider the model of one controller interacting with one worker depicted in Figure 4.8. We demonstrate that the conjunction of the new generated invariant together with the interaction invariant and the local invariants of the components extended with history clocks is strong enough to infer the desired safety property $\Psi \equiv (lc_1 \wedge l_1^1 \rightarrow x \leq y_1)$. The invariants for the components with history clocks are computed precisely as illustrated in Chapter 4 as the sets of reachable states:

$$\begin{aligned} CI(Controller^h) \equiv & (lc_0 \wedge x = h_0 < h_a \wedge h_0 < h_c) \vee \\ & (lc_1 \wedge x \leq h_0 - 4 \wedge x \leq 4 \wedge h_0 < h_a \wedge h_0 < h_c) \vee \\ & (lc_1 \wedge x \leq 4 \wedge x = h_c \leq h_a \leq h_0 - 8) \vee \\ & (lc_2 \wedge x \leq h_0 - 8 \wedge h_a = x \wedge h_0 < h_c) \vee \\ & (lc_2 \wedge x = h_a \wedge h_c = h_a + 4 \leq h_0 - 8) \end{aligned}$$

$$\begin{aligned} CI(Worker_1^h) \equiv & (l_1^1 \wedge y_1 = h_0 < h_{d_1} \wedge h_0 < h_{b_1}) \vee \\ & (l_1^1 \wedge y_1 = h_{d_1} \leq h_{b_1} \leq h_0 - 4) \vee \\ & (l_2^1 \wedge h_{b_1} + 4 \leq y_1 = h_0 < h_{d_1}) \vee \\ & (l_2^1 \wedge y_1 = h_{d_1} \leq h_0 - 4 \wedge h_{b_1} \leq h_{d_1} - 4) \end{aligned}$$

The history clocks inequalities predicate $\mathcal{E}((a \mid b_1), (c \mid d_1))$ is equal to $h_a = h_{b_1} \wedge h_c = h_{d_1}$. By using in addition the interaction invariant described in Section 4.6 and after the elimination of the existential quantifiers in

$$\big(\exists h_a.\exists h_{b_1}.\exists h_c.\exists h_{d_1}.\exists h_0\big)\big(CI(Controller^h) \wedge CI(Worker_1^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma)\big)$$

we obtain the following global invariant GI:

$$\begin{aligned} GI \equiv & (l_1^1 \wedge lc_0 \wedge \ x = y_1) \vee \\ & \big(l_1^1 \wedge lc_1 \wedge (y_1 = x \ \vee \ x + 4 \leq y_1)\big) \vee \\ & \big(l_2^1 \wedge lc_2 \wedge (y_1 = x + 4 \ \vee \ x + 8 \leq y_1)\big). \end{aligned}$$

The relations between x and y_1 that are newly introduced thanks to the history clocks are highlighted by the blue color. As demonstrated in Section 4.6, they could not be deduced from the invariant $CI(Controller) \wedge CI(Worker_1) \wedge II(\gamma)$. At the opposite, the new invariant allows to catch time synchronization between the clocks of the different components and does capture the safety property Ψ.

In case where the interaction set is the union of disjunctive subsets of interactions, the actions history clocks inequalities can have a simpler form.

Proposition 5.4. *If $\gamma = \gamma_1 \cup \gamma_2$ such that $Act(\gamma_1) \cap Act(\gamma_2) = \emptyset$, then $\mathcal{E}(\gamma) \equiv \mathcal{E}(\gamma_1) \wedge \mathcal{E}(\gamma_2)$.*

Proof. We reason by induction on the number of interactions in γ. In case where the size is equal to one, the result is trivial since the set γ contains an only interaction. For the induction step, we use $eq(\alpha)$ and $leq(\alpha, \gamma)$ to denote respectively $\bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j}$ and $\bigwedge_{\substack{a_i \in \alpha \\ a_k \in Act(\gamma \ominus \alpha)}} h_{a_i} \leq h_{a_k}$, the history clocks inequalities $\mathcal{E}(\gamma)$ can

be rewritten as follows:

$$\begin{aligned}
\mathcal{E}(\gamma) &\equiv \bigvee_{\alpha\in\gamma_1} eq(\alpha)\wedge leq(\alpha,\gamma)\wedge\mathcal{E}((\gamma_1\cup\gamma_2)\ominus\alpha)\vee\bigvee_{\alpha\in\gamma_2} eq(\alpha)\wedge leq(\alpha,\gamma)\wedge\mathcal{E}((\gamma_1\cup\gamma_2)\ominus\alpha)\\
&\quad\big(\text{using } \gamma_2\ominus\alpha=\gamma_2 \text{ for } \alpha\notin\gamma_2 \text{ and by induction hypothesis on } \gamma'=\gamma_2\cup(\gamma_1\ominus\alpha)\big)\\
&\equiv \bigvee_{\alpha\in\gamma_1} eq(\alpha)\wedge leq(\alpha,\gamma)\wedge\mathcal{E}(\gamma_1\ominus\alpha)\wedge\mathcal{E}(\gamma_2)\vee\bigvee_{\alpha\in\gamma_2} eq(\alpha)\wedge leq(\alpha,\gamma)\wedge\mathcal{E}(\gamma_1)\wedge\mathcal{E}(\gamma_2\ominus\alpha)\\
&\quad\big(\text{using } \text{ for } i\in\{1,2\}.\,\mathcal{E}(\gamma_i)=\bigvee_{\alpha\in\gamma_i} eq(\alpha)\wedge leq(\alpha,\gamma_i)\wedge\mathcal{E}(\gamma_i\ominus\alpha)\big)\\
&\equiv \mathcal{E}(\gamma_1)\wedge\mathcal{E}(\gamma_2)\wedge\big(\bigvee_{\alpha\in\gamma_1} leq(\alpha,\gamma_2)\vee\bigvee_{\alpha\in\gamma_2} leq(\alpha,\gamma_1)\big)\\
&\quad(\text{using totality of "}\leq\text{" and disjointness of }\gamma_i)\\
&\equiv \mathcal{E}(\gamma_1)\wedge\mathcal{E}(\gamma_2)
\end{aligned}$$

This corollary follows from Proposition 5.4:

Corollary 5.2. *If the interaction model γ has only disjoint interactions, i.e., for any $\alpha_1,\alpha_2\in\gamma$, $\alpha_1\cap\alpha_2\equiv\emptyset$, then $\mathcal{E}(\gamma)\equiv\bigwedge_{\alpha\in\gamma}\Big(\bigwedge_{a_i,a_j\in\alpha} h_{a_i}=h_{a_j}\Big)$.*

For example the two interactions $a\mid b_1$ and $c\mid d_1$ are disjoint, thus this formula is valid: $\mathcal{E}(\{(a\mid b_1),(c\mid d_1)\})\equiv(h_a=h_{b_1})\wedge(h_c=h_{d_1})$.

5.2 History Clocks for Interactions

The history clocks inequalities allow to obtain relations between the clocks of the different components. In case of non conflicting interactions, they are expressed in the conjunctive form and they result in rather a "tight" invariant. However, in case of conflicting interactions, an action may be executed exclusively by one interaction at a given time while the history clocks inequalities have a disjunctive form reflecting an uncertainty about the effective participation of the action in one of the interactions which share it. Therefore, in case of conflicting interactions, the inequalities are relatively "loose". The presence of conflicts is capitalized on bringing up new invariants. In fact, when an action is executed as part of an interaction, it should be again enabled in order to give the possibility for another interaction sharing it to be executed. In some cases, a minimum time elapse is required between two executive occurrences of the same action. In this case, a minimum separation time is imposed between the execution of two conflicting interactions.

In order to capture this, we make use of history clocks for interactions which behave in a way similar to the history clocks for actions. To each interaction, we introduce a history clock which is reset whenever it is executed. This allows to reason at interactions level, besides the relations that can be derived between interactions history clocks and actions history clocks.

Definition 5.3 (System with Interaction History Clocks)**.**
Given a timed system $\|_\gamma B_i$, its extension with history clocks for interactions is the timed system $B^* \|_{\gamma^h} B_i^h$ where:

- *B^* is an auxiliary component $(\{l^*\}, A_\gamma, \mathcal{H}_\gamma, T, (l^* \mapsto \mathbf{true}), (l^*, \mathbf{true}))$ where:*
  - *$A_\gamma = \{a_\alpha \mid \alpha \in \gamma\}$ is the set of actions.*
  - *$\mathcal{H}_\gamma = \{h_\alpha \mid \alpha \in \gamma\}$ is the set of interaction history clocks.*
  - *$T = \{(l^*, (a_\alpha, \mathbf{true}, \{h_\alpha\}), l^*) \mid \alpha \in \gamma\}$ is the set of transitions.*
- *$\gamma^h = \{(a_\alpha \mid \alpha) \mid \alpha \in \gamma\}$ where $(a_\alpha \mid \alpha)$ denotes $\{a_\alpha\} \cup \{a \mid a \in \alpha\}$.*

Since the interaction history clocks do not affect the behavior of the system, any invariant of $B^* \|_{\gamma^h} B_i^h$ corresponds to an invariant of $\|_\gamma B_i$ obtained through existential quantifiers elimination.

Proposition 5.5.

1. *If Φ^h is an invariant of $B^* \|_{\gamma^h} B_i^h$, then $\Phi = \exists \mathcal{H}_A \exists \mathcal{H}_\gamma . \Phi^h$ is an invariant of $\|_\gamma B_i$.*
2. *If Φ^h is an invariant of $B^* \|_{\gamma^h} B_i^h$ and Ψ^h is an inductive predicate of $B^* \|_{\gamma^h} B_i^h$ expressed on history clocks for actions and interactions $\mathcal{H}_\gamma \cup \mathcal{H}_A$, then $\Phi = \exists \mathcal{H}_A \exists \mathcal{H}_\gamma . (\Phi^h \wedge \Psi^h)$ is an invariant of $\|_\gamma B_i$.*

Proof. The proofs of (1) and (2) use similar arguments as the proofs of propositions 5.1 and 5.3 respectively.

5.2.1 Invariants for Interaction History Clocks

The interactions history clocks serve to introduce new invariants reflecting some constraints on the execution of interactions. Given two interactions sharing an action a, if one of them is executed, then the other should wait for the action a to be again enabled. This characteristic is encoded in a new invariant, the so-called *separation constraints*.

Definition 5.4 (Separation Constraints for Interaction History Clocks)**.** *Given an interaction set γ, the separation constraints $\mathcal{S}(\gamma)$ predicate is defined as follows:*

$$\mathcal{S}(\gamma) \equiv \bigwedge_{a \in Act(\gamma)} \bigwedge_{\substack{\alpha \neq \beta \in \gamma \\ a \in \alpha \cap \beta}} \mid h_\alpha - h_\beta \mid \geq k_a$$

where $\mid h \mid$ denotes the absolute value of h and k_a is a constant computed locally on the component containing action a. It represents the minimum time elapse between two consecutive occurrences of a.

As an example to illustrate the usage of separation constraints, we consider the system composed of the controller interacting with two workers. Interactions $a|b_1$ and $a|b_2$ are conflicting on action a, whereas interactions $c|d_1$ and $c|d_2$ are conflicting on action c.

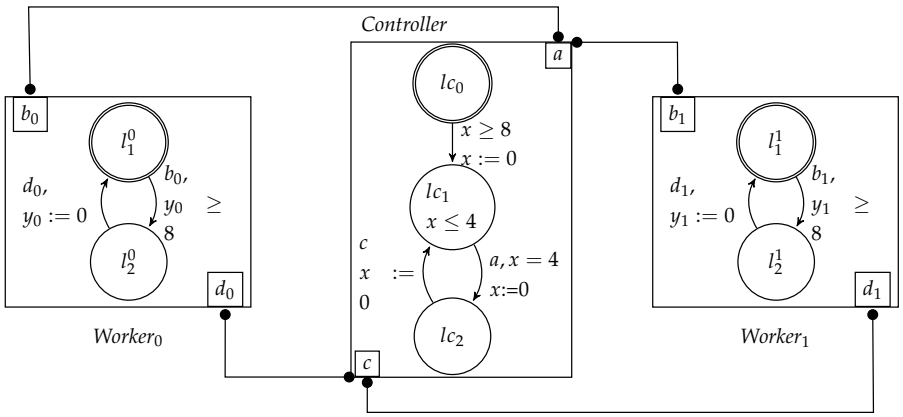


Figure 5.1: A timed System

Both actions a and c of the controller component need 4 time units between two executions. It results that constants k_a and k_c are equal to 4. The resulting separation constraints are expressed as follows:

$$\mathcal{S}((a \mid b_i)_i, (c \mid d_i)_i) \equiv \bigwedge_{i \neq j} |h_{a|b_i} - h_{a|b_j}| \geq 4 \wedge \bigwedge_{i \neq j} |h_{c|d_i} - h_{c|d_j}| \geq 4$$

On the Computation of the Separation Constants

Let B be a component with actions set A and let a be an action in A. The constant k_a is the minimal time required between two consecutive occurrences of a. Formally, it is defined as follows:

$$k_a = \min_{i \geq 1}(occ(i+1, a) - occ(i, a))$$

The value $occ(i,a)$ denotes the minimum time elapse until the i-th occurrence of a is reached, else it is equal to ∞ if the number of occurences is strictly smaller than i. Its computation was detailed in [Cou91]. In the following, we show a dynamic programming solution:

$$occ(i,a) = \min_{s \in R} mt(s_0, s, a, i)$$

where s_0 is the initial symbolic state of B^e, B^e is the component B enriched with the auxiliary clock h_0 measuring the time elapse since the beginning. Given two symbolic states s and f and a transition labeled with a, the value of $mt(s,f,a,1)$ is computed as follows:

$$mt(s,f,a,1) = \begin{cases} msol\big(\zeta(f)\big) & \text{if } s \xrightarrow{a} f \\ \infty & \text{otherwise} \end{cases}$$

$$mt(s,f,a,p) = \min_{k \in Reach(B^e), b \in A} \big(mt(s,k,b,p-1) + mt(k,f,a,1)\big)$$

where $msol\big(\zeta(s)\big)$ is the minimal solution over h_0 to the constraints in the zone of the symbolic state s.

5.2.2 Strengthening the Global Invariant

In order to allow gluing together the history clocks inequalities defined on the actions history clocks, and the separation constraints defined on the interactions history clocks, we propose another formulation for the separations taking into account the valid relations between these two kinds of history clocks.

Proposition 5.6. *Let*

$$\mathcal{S}^*(\gamma) \equiv \bigwedge_{a \in Act(\gamma)} \bigwedge_{\substack{\alpha \neq \beta \in \gamma \\ a \in \alpha \cap \beta}} (h_a \leq h_\alpha \leq h_\beta - k_a \vee h_a \leq h_\beta \leq h_\alpha - k_a)$$

We have that:

1. *$\mathcal{S}^*(\gamma)$ is an inductive predicate of $B^* \|_{\gamma^h} B_i^h$.*
2. *The equivalence $\mathcal{S}(\gamma) \equiv \exists \mathcal{H}_A . \mathcal{S}^*(\gamma)$ is valid formula.*

Proof.
1. We reason by induction on the execution sequence. We assume that $\mathcal{S}^*(\gamma)$ is satisfied at an arbitrary state s of the system and proceed to prove that it is satisfied

by all of its successor states. There are two types of successors: time successors and discrete successors. As for time successors, all the relations between the clocks are preserved as they do all increase at the same rate. We propose to prove that $\mathcal{S}^*(\gamma)$ is preserved for any discrete successor of s.

We fix arbitrarily an action a, and two interactions α and β competing on a and prove that the following predicate is inductive:

$$S(a,\alpha,\beta) \equiv (h_a \leq h_\alpha \leq h_\beta - k_a \vee h_a \leq h_\beta \leq h_\alpha - k_a)$$

When a state s' is reached from s through an interaction different from α and β, h_a is reset whereas the values of h_α and h_β are not affected thus $S(a,\alpha,\beta)$ is preserved. If we consider that β interaction is executed, then we distinguish two cases:

- $h_a \leq h_\beta \leq h_\alpha - k_a$ is satisfied at s state: during the execution of β, h_a and h_β are reset whereas h_α is changed. It results that the predicate remains satisfied at the successor state.
- $h_a \leq h_\alpha \leq h_\beta - k_a$ is satisfied at s state: Given that at least k_a time units are needed between two consecutive executions of a action, we deduce that before the execution of β toward s' state, the value of h_a is bigger than k_a. Therefore h_α is also bigger than k_a. At the moment of execution, clocks h_a and h_β are reset. It results that at s', $h_a = h_\alpha = 0$ and the predicate $h_a \leq h_\beta \leq h_\alpha - k_a$ is valid at s' state.

In a symmetric manner, we can show that the predicate is preserved if α is executed.

2. proof that the equivalence between $\mathcal{S}(\gamma)$ and $\exists \mathcal{H}_A.\mathcal{S}^*(\gamma)$ is a valid:

$$\begin{aligned}
\mathcal{S}^*(\gamma) &\equiv \bigwedge_{a\in Act(\gamma)} \bigwedge_{\substack{\alpha\neq\beta\in\gamma \\ a\in\alpha\cap\beta}} (\mid h_\alpha - h_\beta \mid \geq k_a \wedge h_a \leq h_\alpha \wedge h_a \leq h_\beta) \\
&\equiv \bigwedge_{a\in Act(\gamma)} \bigwedge_{\substack{\alpha\neq\beta\in\gamma \\ a\in\alpha\cap\beta}} (\mid h_\alpha - h_\beta \mid \geq k_a) \wedge \bigwedge_{\substack{\alpha\neq\beta\in\gamma \\ a\in\alpha\cap\beta}} (h_a \leq h_\alpha \wedge h_a \leq h_\beta) \\
&\equiv \mathcal{S}(\gamma) \wedge \bigwedge_{a\in Act(\gamma)} \bigwedge_{\substack{\alpha\neq\beta\in\gamma \\ a\in\alpha\cap\beta}} (h_a \leq h_\alpha \wedge h_a \leq h_\beta)
\end{aligned}$$

The formula $\mathcal{S}(\gamma)$ is expressed over the interaction history clocks whereas the components invariants $CI(B_i^h)$ are expressed over actions history clocks. In order to glue them together in a relevant way, we need to introduce relations between interactions and actions history clocks.

Definition 5.5 ($\mathcal{E}^*$)**.**
Given an interaction set γ, the predicate $\mathcal{E}^*(\gamma)$ is defined as follows:

$$\mathcal{E}^*(\gamma) \equiv \bigwedge_{a \in Act(\gamma)} h_a = \min_{\alpha \in \gamma, a \in \alpha} h_\alpha.$$

Using the same reasoning as in Proposition 5.2, it can be shown that $\mathcal{E}^*(\gamma)$ is an inductive predicate of the system $B^* \|_{\gamma^h} B_i^h$ enriched with history clocks.

Proposition 5.7.

1. *$\mathcal{E}^*(\gamma)$ is an inductive predicate of $B^* \|_{\gamma^h} B_i^h$.*

2. *The predicate $\exists \mathcal{H}_\gamma . \mathcal{E}^*(\gamma)$ is equivalent to $\mathcal{E}(\gamma)$.*

Proof.
1. If $\mathcal{E}^*(\gamma)$ is satisfied at a state s of the system $B^* \|_{\gamma^h} B_i^h$, then it is satisfied by all the time successors since the clocks advance at the same rate. We consider arbitrarily an action a and show that the predicate $h_a = \min_{\alpha \in \gamma, a \in \alpha} h_\alpha$ is inductive. If a discrete state is reached from s through an interaction which does not contain a action, then the predicate is preserved. Else, if it is reached through an interaction α containing a, then the history clocks h_a and h_α are reset simultaneously and remain equal whereas the history clocks relative to the other interaction are not affected and thus remain bigger than h_α.

2. We consider that the set of interactions is $\gamma = \{\alpha_1, \alpha_2, ..., \alpha_m\}$:

$$\exists \mathcal{H}_\gamma.\mathcal{E}^*(\gamma) \equiv \exists \mathcal{H}_\gamma. \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec ... \prec \alpha_{k_m}} \big(h_{\alpha_{k_1}} \leq h_{\alpha_{k_2}} \leq ... \leq h_{\alpha_{k_m}} \wedge \mathcal{E}^*(\gamma)\big)$$

(disjointness on the possible orderings $\prec$ on interactions)

$$\equiv \exists \mathcal{H}_\gamma. \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec ... \prec \alpha_{k_m}} \big(h_{\alpha_{k_1}} \leq h_{\alpha_{k_2}} \leq ... \leq h_{\alpha_{k_m}} \wedge \bigwedge_{a \in \alpha_{k_1}} (h_a = h_{\alpha_{k_1}}) \wedge \bigwedge_{a \in \alpha_{k_2} \setminus \alpha_{k_1}} (h_a = h_{\alpha_{k_2}}) \wedge ... \bigwedge_{a \in \alpha_{k_m} \setminus \alpha_{k_1} ... \alpha_{k_{m-1}}} (h_a = h_{\alpha_{k_m}})\big)$$

(by expansion of $\mathcal{E}^*(\gamma)$ along the chosen order)

$$\equiv \exists \mathcal{H}_\gamma. \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec ... \prec \alpha_{k_m}} \big(h_{\alpha_{k_1}} \leq h_{\alpha_{k_2}} \leq ... \leq h_{\alpha_{k_m}} \wedge \bigwedge_{\ell=1}^{m} \bigwedge_{a \in \alpha_{k_\ell} \setminus \alpha_{k_1} ... \alpha_{k_{\ell-1}}} (h_a = h_{\alpha_{k_\ell}})\big)$$

$$\equiv \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec ... \prec \alpha_{k_m}} \exists \mathcal{H}_\gamma. \big(h_{\alpha_{k_1}} \leq h_{\alpha_{k_2}} \leq ... \leq h_{\alpha_{k_m}} \wedge \bigwedge_{\ell=1}^{m} \bigwedge_{a \in \alpha_{k_\ell} \setminus \alpha_{k_1} ... \alpha_{k_{\ell-1}}} (h_a = h_{\alpha_{k_\ell}})\big)$$

(by distribution the existential quantifiers over the disjunction elements)

$$\equiv \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec ... \prec \alpha_{k_m}} \bigwedge_{\ell=1}^{m} \bigwedge_{\substack{a_i, a_j \in \alpha_{k_\ell} \setminus \alpha_{k_1} ... \alpha_{k_{\ell-1}} \\ a_k \notin \alpha_{k_1} ... \alpha_{k_\ell}}} (h_{a_i} = h_{a_j} \leq h_{a_k})$$

$$\equiv \mathcal{E}(\gamma)$$

From Propositions 5.5, 5.6 and 5.7, it results that $\exists \mathcal{H}_A \exists \mathcal{H}_\gamma.(\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma))$ is an invariant of $\|_\gamma B_i$. In general, this invariant is stronger than $\exists \mathcal{H}_A.(\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma))$ in cases there are interactions conflicting on an action a whose constant k_a is strictly positive exist.

Example 5.2. To show the enhancement brought by the invariant generated using separation constraints, we reconsider the example with two workers. The subformula which is relative to the history clocks here is the conjunction of $\mathcal{E}^*$ with $\mathcal{S}$. The interaction invariant is:

$$II(\gamma) \equiv (l_1^1 \vee lc_1 \vee lc_2) \wedge (l_1^2 \vee lc_1 \vee lc_2) \wedge (lc_2 \vee l_1^1 \vee l_1^2) \wedge (lc_0 \vee lc_1 \vee l_2^1 \vee l_2^2)$$

The components invariants are:

$$\begin{aligned} CI(Controller^h) \equiv & (lc_0 \wedge x = h_0 \wedge h_0 < h_a \wedge h_0 < h_c) \vee \\ & (lc_1 \wedge x \leq h_0 - 8 \wedge x \leq 4 \wedge h_0 < h_a \wedge h_0 < h_c) \vee \\ & (lc_1 \wedge x \leq 4 \wedge x = h_c \leq h_a \leq h_0 - 12) \vee \\ & (lc_2 \wedge x \leq h_0 - 12 \wedge h_a = x \wedge h_0 < h_c) \vee \\ & (lc_2 \wedge x = h_a \wedge h_c = h_a + 4 \leq h_0 - 12) \end{aligned}$$

$$\begin{aligned} CI(Worker_i^h) \equiv & (l_1^i \wedge y_i = h_0 \wedge h_0 < h_{d_i} \wedge h_0 < h_{b_i}) \vee \\ & (l_1^i \wedge y_i = h_{d_i} \leq h_{b_i} \leq h_0 - 8) \vee \\ & (l_2^i \wedge y_i \geq h_{b_i} + 8 \leq h_0 < h_{d_i}) \vee \\ & (l_2^i \wedge y_i = h_{d_i} \leq h_0 - 8 \wedge h_{b_i} \leq h_{d_i} - 8) \end{aligned}$$

The inequalities for action and interaction history clocks are:

$$\begin{aligned} \mathcal{E}^*(\gamma) \equiv & (h_{b_1} = h_{a|b_1}) \wedge (h_{b_2} = h_{a|b_2}) \wedge \ (h_a = \min_{i=1,2}(h_{a|b_i})) \wedge \\ & (h_{d_1} = h_{c|d_1}) \wedge (h_{d_2} = h_{c|d_2}) \wedge \ (h_c = \min_{i=1,2}(h_{c|d_i})) \end{aligned}$$

By recalling the expression of $\mathcal{S}(\gamma)$, we obtain that:

$$\mathcal{S}(\gamma) \ \equiv (|h_{b_2} - h_{b_1}| \geq 4 \ \wedge |h_{d_2} - h_{d_1}| \geq 4)$$

and thus, after simplification and quantifier elimination in

$$\exists \mathcal{H}_A \exists \mathcal{H}_\gamma . (CI(Controller^h) \wedge \bigwedge_i CI(Worker_i^h) \wedge II(\gamma) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma))$$

we obtain the following global invariant GI:

$$\begin{aligned} GI \equiv & (l_1^1 \wedge l_1^2 \wedge lc_0 \wedge \ x = y_1 = y_2) \vee \\ & (l_1^1 \wedge l_1^2 \wedge lc_1 \wedge x \leq 4 \wedge (y_1 = y_2 \geq x + 8 \vee \\ & \qquad (y_1 = x \ \wedge y_2 - y_1 \geq 4) \vee \\ & \qquad (y_1 \geq x + 8 \wedge y_1 - y_2 \geq 8) \vee \\ & \qquad (y_2 = x \ \wedge y_1 - y_2 \geq 4) \vee \\ & \qquad (y_2 \geq x + 8 \wedge y_2 - y_1 \geq 8))) \vee \\ & (l_2^1 \wedge l_1^2 \wedge lc_2 \wedge y_1 \geq x + 8 \wedge ((y_2 \geq x + 4 \wedge |y_1 - y_2| \geq 4) \vee \\ & \qquad y_2 \geq x + 12)) \vee \\ & (l_1^1 \wedge l_2^2 \wedge lc_2 \wedge y_2 \geq x + 8 \wedge ((y_1 \geq x + 4 \wedge |y_1 - y_2| \geq 4) \vee \\ & \qquad y_1 \geq x + 12)) \end{aligned}$$

In this expression, we emphasize with the blue color the new relations induced from the use of separation constraints. They make the global invariant GI strong enough to prove that the property of interest $\Psi \equiv l_1^0 \wedge l_1^1 \wedge lc_0 \Rightarrow (y_1 - x \geq 4 \vee y_2 - x \geq 4)$ is satisfied by the system.

5.3 Action Occurrence Invariants

The clock h_0 measures the amount of time elapsed since the start moment. It is never reset nor is there a guard on its value in the transitions. All the clocks advance at the same, and so does h_0 which is common to all the components of the system. Besides, the clock h_0 serves to deduce which actions have been executed so far. The initialization of the history clocks helps to detect this information. All the action history clocks are initialized to be strictly bigger than 0 whereas the inner clocks of the components and the clock h_0 are initialized to 0. It results that when an action occurs, the related history clock is reset while the value of h_0 is positive. Therefore, an action history clock is strictly bigger than h_0 if and only if its action has not been executed at all.
We mention that if the history clocks were initialized to be equal to 0 at the start date, the global invariant would inaccurately allow to reflect, for instance, that all the actions have occurred at the start date, which is generally inaccurate.

Refining Relations between Conflicting Interactions The initialization of the action and interaction history clocks and the clock h_0 measuring the time elapsed since the beginning allows to deduce whether or not an action or interaction has been executed. In fact, the predicate $h_0 < h_\alpha$ intuitively means that interaction α has not been executed yet, whereas the clock h_0 is bigger or equal to h_α only if α has been executed at least once. In the following, we show a new family of invariants related to conflicting interactions. We consider an action a from a component B that is conflicting and two interactions α_1 and α_2 which share it. Intuitively, we know that if both of these interactions have been executed, meaning that action a has been executed at least twice, there must be at least one action preceding a that has been executed. This allows the definition of a new invariant expressed formally as follows:

$$h_{\alpha_1} \leq h_0 \wedge h_{\alpha_2} \leq h_0 \Rightarrow \bigvee_{a' \in Prec(a)} h_{a'} \leq h_0$$

The set $Prec(a)$ contains the actions within component B that can be immediate predecessors of action a, i.e $Prec(a) = \{a' \in A \mid \exists l, l', l'' \in L. l \xrightarrow{a'} l', l' \xrightarrow{a} l''\}$.
We note that this new invariant is implied by the component invariant $CI(B)$ if a is not enabled at the initial state. In this case, any state where a can be executed should be reached by a non-empty execution sequence. Therefore, whenever a has been executed, at least one of its preceding actions has been executed as well. This is necessarily considered during the computation of the component invariant. We deduce that if a is not enabled at the initial state, $CI(B)$ implies the following assertion:

$$h_a \leq h_0 \Rightarrow \bigvee_{a' \in Prec(a)} h_{a'} \leq h_0$$

At the opposite case, if a is an action enabled at the initial state, then it can be the first action to be executed without need for precedency of any other action. In such a case, a is executed once while none of its preceding action has been executed and the new proposed invariant is relevant. Besides, the action history clock h_a reflects the last occurrence of action a and gives no clue of to distinguish whether a was executed once or more. This can be deduced, if a is conflicting, from the history clocks of the interactions sharing it.

Example 5.3. We consider the component depicted in Figure 5.2 where the port a is shared between two interactions α_1 and α_2. The following predicate is an invariant of the system:

$$h_{\alpha_1} \leq h_0 \wedge h_{\alpha_2} \leq h_0 \Rightarrow h_b \leq h_0$$

5.4 Component Invariants Revisited

In some cases, for example when the component is untimed, the computation of the local invariant is itself costly. Even in absence of original component clocks, all the possible orders of the history clocks at each of the locations are needed. In such a case, the number of zones in the reachability graph are likely to be large. In some way, the existence of inner component clocks does constrain the dynamics of the component whereas their absence can result in an exponentially big number of possible history clocks orders. It results that the reachability graph computation method is unsuitable for the originally non timed components, which are yet extended with history clocks, and is rather unpractical. In the following of this section, we show a method for computing a shorter component invariant in case where the original component is untimed.

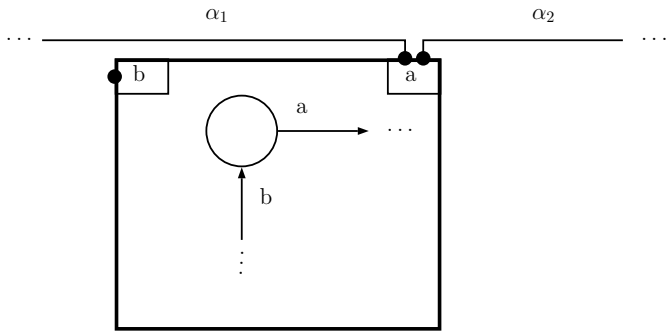


Figure 5.2: A timed component with action a involved in interactions α_1 and α_2

Example 5.4. We consider the untimed component illustrated in Figure 5.3 (left) and its extension with history clocks (right). The entire zone graph reachable from (l_0, ζ_0), where $\zeta_0 = (h_0 = 0, h_{a,b,c} > 0)$, contains 16 symbolic states. It results that the component invariant is expressed as a disjunction of 16 terms, 9 of them correspond to location l_0 while the seven others correspond to location l_1.

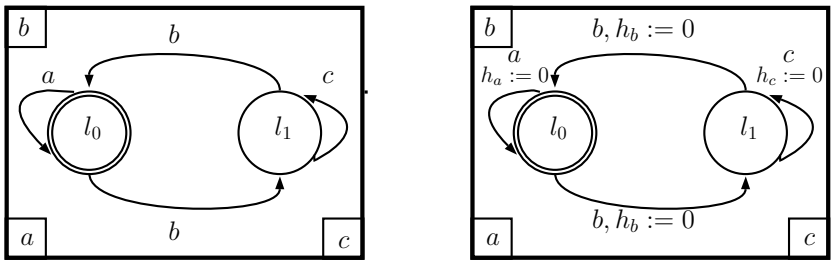


Figure 5.3: An untimed component and its extension with history clocks

The starting point of our method is the fact that the orders of actions for untimed automata have elegant and compact encodings as regular expressions. More concretely, given an untimed component $B = (L, A, T)$, we show how to automatically compute an invariant describing the relations between the history clocks of B^h from the language accepted by B, known that every order on transitions is equivalent to an order on history clocks. The main feature that we exploit is that for the computation of the component invariant, only the last occurrence of each

action is relevant. Therefore, for a given location, it is safe to simplify the regular expressions relative to the accepted language by removing away all but the last occurrences of the different actions. We note that regular expressions converted to a restricted form offer the possibility to generate constraints on actions history clocks that are less numerous than the orders induced by the reachability graph. The method that we propose to generate the set of actions history clocks orders at a given location l follows three main steps:

1. construct the regular expression E_ℓ representing the language accepted by B at location ℓ,

2. abstract E_ℓ with respect to the *last occurrence retention* operation towards a *restricted* form $E_\ell^\sharp = \sum_i e_i^\sharp$ where every action appears at most once in $e_i^\sharp$, and where no nested *-operators remain.

3. generate from every $e_i^\sharp$ a characteristic formula on history clocks $\phi(e_i^\sharp)$, such that the assertion $\ell \Rightarrow \vee_i \phi(e_i^\sharp)$ is an invariant for the component B .

A basic theorem in formal language theory states that every automata can be effectively converted into a regular expression and several algorithms have been proposed to achieve it [HMU03]. In this dissertation, we don't give the details of this conversion and we directly go into the second step. The restricted form of the regular expression is achieved by use of two simplification rules:

Rule 1 [Last Occurrence Retention]:	$E \cdot a \longrightarrow (E \setminus a) \cdot a$
Rule 2 [Back-unfolding]:	$E^* \longrightarrow (E^* \cdot E) + \varepsilon$

Figure 5.4: Simplification Rules

Rule 1 keeps only the last occurrence of the trailing action a in the regular expression of the form $E \setminus a$. The "$\setminus$" denotes a syntactic *elimination operator* defined structurally on regular expressions as follows. Let a and x be two symbols

and E, E_1 and E_2 be arbitrary regular expressions.

$$\begin{aligned}
\varepsilon \setminus a &= \varepsilon \\
x \setminus a &= \begin{cases} \varepsilon \text{ if } x = a \\ x \text{ if } x \neq a \end{cases} \\
(E_1 + E_2) \setminus a &= (E_1 \setminus a) + (E_2 \setminus a) \\
(E_1.E_2) \setminus a &= (E_1 \setminus a).(E_2 \setminus a) \\
E^* \setminus a &= (E \setminus a)^*
\end{aligned}$$

Rule 2 frugally unfolds *-expressions once. By using this rule and basic processing of regular expressions, further simplification occasions for Rule 1 are allowed.

Example 5.5. We again consider the example depicted in Figure 5.3. The language accepted at l_1 location is defined as $(a + bc^*b)^*bc^*$. This expression is progressively converted into the restricted form following these steps:

$$\begin{aligned}
(a + bc^*b)^*bc^* &\rightsquigarrow (a + c^*)^*bc^* && \text{(by Rule 1)} \\
&\equiv (a + c^*)^*b(c^*c + \varepsilon) && \text{(by Rule 2)} \\
&\equiv (a + c^*)^*bc^*c + (a + c^*)^*b && \text{(by splitting the last +)} \\
&\rightsquigarrow (a + \varepsilon)^*bc + (a + c^*)^*b && \text{(by Rule 1)} \\
&\equiv a^*bc + (a + c)^*b && \text{(by standard transformation)}
\end{aligned}$$

In this example, we applied the iterative strategy consisting of alternating the following two operations (1) choosing symbols from right to left and applying **Rule** 1 as long as possible and then (2) applying **Rule** 2 to unfold the rightmost *-expression and split the incoming +. This strategy always terminates with regular expressions in the restricted form. In some way, **Rule** 2 expands the expressions so that **Rule** 1 can be applied to eliminate the symbols repetitions.
As for the third step, orders on actions are inferred from the regular expression under restricted form $e^\sharp$. These orders are formulated as follows, by use of history clocks. They represent the possible orders on the set of actions, encoded by the regular expression.

$$\phi(e^\sharp) \equiv \bigvee_{\substack{a_1 \ldots a_n \in L(e^\sharp) \\ \text{distinct } a_1,\ldots,a_n}} \left(h_0 \geq h_{a_1} \geq \ldots \geq h_{a_n} \wedge \bigwedge_{c \neq a_1,\ldots,a_n} h_c > h_0\right)$$

where $L(e^\sharp)$ is the language of $e^\sharp$.
Since only words with distinct symbols are considered, they are finitely many and the disjunction is also finite. Intuitively, this formula if obtained by ordering the actions following their appearances in the given string, while the remaining actions of the component that do not appear in the string are supposed to not be executed, thus their action history clocks are bigger than h_0, the clock measuring the time elapsed since the beginning. As an illustration, we denote here by $e^\sharp$ the regular expression under the restricted form $a^*bc + (a+c)^*b$ obtained for Example 5.5. We show at first a direct encoding of the orders resulting from the finite words on which $\phi(e^\sharp)$ is built: abc and bc (from a^*bc) and acb, cab, cb, ab, b from $(a+c)^*b$.

$$
\begin{array}{ll}
(h_0 \geq h_a \geq h_b \geq h_c) \vee (h_a > h_0 \geq h_b \geq h_c) \vee & (\text{corresponding. to } abc, \text{resp. } bc) \\
(h_0 \geq h_a \geq h_c \geq h_b) \vee (h_a > h_0 \geq h_c \geq h_b) \vee & (\text{corr. to } acb, \text{resp. } cb) \\
(h_0 \geq h_c \geq h_a \geq h_b) \vee (h_c > h_0 \geq h_a \geq h_b) \vee & (\text{corr. to } cab, \text{resp. } ab) \\
(h_0 \geq h_b \wedge h_c, h_a > h_0) & (\text{corr. to } b)
\end{array}
$$

Later, a more efficient translation from a given restricted regular expression to the orders on action history clocks is detained, but we show before that such encodings are invariants. The inequalities in $\phi(e^\sharp)$ reflect precisely the order in which the last occurrence of each action has taken place. If it never occurred, its history clock is bigger than h_0.

Proposition 5.8. *Let B be an untimed component, E_l the regular expression characterizing the language accepted by B at location l, and $E_l^\sharp$ be the result of applying the simplification rules on E_l. The predicate $\bigvee_l (l \wedge \phi(E_l^\sharp))$ is an invariant of the component B^h extended with history clocks.*

Proof. (a sketch)
Since the original component is untimed, the computation of its component invariant aims exclusively at finding the order over the last occurrences of the action history clocks when reaching the different locations. For a given location l, the language of the last occurrences of actions is preserved by E_l and $E_l^\sharp$ along the simplification rules. Besides, for every regular expression $e^\sharp$ of $E_l^\sharp$ under the restricted form, containing each action at most once, the following property is valid: the language $L(e^\sharp)$ of $e^\sharp$ includes every word w_{last} resulting from the elimination of all but the last occurrences from a word w in $L(e^\sharp)$. Consequently, all the last occurrence words w_{last} can be enumerated by taking all the accepted word of $L(e^\sharp)$ having necessarily distinct symbols. Finally, we note that the inequalities in $\phi(E_l^\sharp)$ encode exactly all the possible words corresponding to traces of component B^h which reach l location.

Obtaining more Compact Invariants from $E_l^\sharp$

The regular expression obtained under the restricted form can be further exploited in order to obtain more compact orderings than those in $\phi(E_l^\sharp)$. For illustration purpose, we consider the regular expression $e^\sharp = (b_1 + ... + b_m)^* a_1 ... a_n$, where the actions $a_1, \ldots, a_n, b_1, \ldots, b_m$ are distinct. This expression reflects that the actions $a_1, ..., a_n$ are *mandatory*, meaning that they necessarily occur in the precise order from a_1 to a_n and their corresponding history clocks are unavoidably smaller than h_0. At the opposite, the actions $b_1, b_2 \ldots, b_m$ are *optional*. Each one of them can eventually occur, thus their history clocks can be smaller or bigger than h_0. Besides, if some of them occur, then the order on their executions in unconstrained. Nevertheless, in all cases, their history clocks are necessarily bigger than the history clock of the first executed mandatory action a_1. We can also interpret that the actions of B^h which do not appear in the regular expression have never occurred. As a result, their history clocks have never been reset, hence they are strictly bigger than h_0. All in all, the following formula summarizes compactly the union of all the possible orders at location l induced from the regular expression $E_l^\sharp$:

$$h_{b_1} \geq h_{a_1} \wedge ... \wedge h_{b_m} \geq h_{a_1} \wedge h_0 \geq h_{a_1} \geq ... \geq h_{a_n} \wedge \bigwedge_{c \neq a_i, b_j} h_c > h_0.$$

We notice that for this example, the obtained behavior is linear on the size of the regular expression. At the opposite, the number of encoded strings is exponential with respect to the number of the optional b actions, since their appearances and mutual orders are unconstrained. We note that the above construction can be conveniently generalized for any arbitrary regular expression following the restricted form. The resulting formula is polynomial and at worse quadratic with respect to the size of the given restricted regular expression.

Example 5.6. Following the above described approach, the regular expression in the restricted form $a^*bc + (a + c)^*b$ translates into the predicate

$$\phi_{comp}(E_l^\sharp) \equiv (h_0 \geq h_b \geq h_c \wedge h_a \geq h_b) \vee (h_0 \geq h_b \wedge h_a \geq h_b \wedge h_c \geq h_b)$$

This expression is drastically smaller, yet logically equivalent to the disjunction of the 7 distinct terms corresponding to the symbolic zones reached at l_1 as initially detailed in Example 5.5. In the following, we give the correspondence between the words in Example 5.5 and their representing terms in the compact formula $\phi_{comp}(E_l^\sharp)$.

- The term $(h_0 \geq h_b \geq h_c \wedge h_a \geq h_b)$ is nothing but the union of the terms corresponding to words abc and bc.

- The term $(h_0 \geq h_b \wedge h_a \geq h_b \wedge h_c \geq h_b)$ is the union of the terms corresponding to the words *acb*, *cb*, *cab ab* and *b*, ending all with *b* action.

To summarize, we presented in this subsection a heuristic applicable to originally untimed components. It allows to automatically generate an invariant for the components extended with history clocks havong the advantage of being enough compact to be handled by existing SMT-Solvers. The use of regular expressions allows for the avoidance of the computation of the whole reachability graph for such components, yielding generally a large number of symbolic states .

5.5 Summary

In this chapter, a method for fully compositional invariant generation for timed systems is detailed. It lays upon the use of auxiliary history clocks in order to track the time synchronization between the components. As later shown in Chapter 9, this invariant is strong enough to prove several properties for different systems. Nevertheless, the above method is based on an over-approximation of the reachable states set, hence false positives may raise in some cases and they appear particularly in heavily non-deterministic systems.

To remedy this, we completed our compositional verification method with a counterexample-based invariant refinement module analyzing iteratively the generated counterexamples. A detailed presentation of this extension is given in the next chapter.

Chapter 6

Counterexample-Based Invariant Refinement

> *It is impossible to overestimate the importance of the counterexample feature. The counterexamples are invaluable in debugging complex systems.*
>
> Edmund Clarke, *The Birth of Model Checking* [Cla08]

Contents

In order to avoid the state-space explosion inherent to model-checking timed systems, we make twofold our verification approach. First, as explained in Chapter 5, we propose a fully compositional method for generating a global invariant of the

system, over-approximating the set of reachable global states. Our compositional invariant generation approach lays upon the combination of local invariants of the different components, and additional invariants obtained from the analysis of the interaction structure of the system. They allow to capture the time synchronization between the components in a fully compositional manner. We find their conjunction quite compromising in terms of tightness and complexity, but the method is still incomplete and spurious counterexamples may appear. In fact, if the safety property is satisfied by the global invariant, it is also satisfied by the system. However, if a configuration satisfies the global invariant and violates the desired safety property, then it may be the outcome of some behavior in the over-approximation which does not belong to the original model.

In the second stage, which is the main subject of this chapter, we proceed to the iterative analysis of the generated counterexamples in order to eliminate the false positives. The suspected counterexamples are analyzed by use of a backward search algorithm. If a counterexample cannot be reached from the initial state, then the global invariant is restricted such that its configuration is eliminated. The satisfiability checking-refining loop is iteratively performed until the cumulatively refined global invariant does not contain any state violating the safety property, in which case the system is confirmed to be safe, or until a counterexample is proven to be reachable from the initial state, thus valid. In the latter case, we deduce that the safety property is not satisfied.

6.1 The Algorithm

In our framework, the generated counterexamples precise the location of each component, together with the clock valuations. They are generated by the SAT-solver as configurations that refute the validity of $GI \Rightarrow \Psi$. After analysis, the refuted counterexamples serve to refine the invariant GI. To implement it and in order to maneuver efficiently the counterexamples and their predecessors, we extend the notion of symbolic state from components to systems of parallel composition. The global location of a system is a n-tuple containing one location of each component and the zone of a global symbolic state is the conjunction of constraints relating the different components clocks.

The algorithm is shown in Figure 6.1. The backward computation starts from a generalized counterexample and computes iteratively its preimage, resulting at

Input: The global invariant $GI \equiv \bigwedge CI(B_i^h) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma) \wedge II(\gamma)$
The property Ψ

```
1  V ← ∅ ;
2  while GI ∧ ¬Ψ is satisfiable do
3  |  Let θ a solution of GI ∧ ¬Ψ ;
4  |  Let (l^θ, ζ^θ) ← generalize(θ, Ψ) ;
5  |  Let P ← {(l^θ, ζ^θ)} ;
6  |  while P ∩ Init = ∅ and P ≠ ∅ do
7  |  |  V ← V ∪ P ;
8  |  |  P ← pre(P) \ V ;
9  |  end
10 |  if P ∩ Init ≠ ∅ then
11 |  |  stop ;
12 |  |
13 |  else
14 |  |  GI ← GI ∧ ¬(at(l^θ) ∧ ζ^θ) ;
15 |  |
16 |  end
17 end
18 Ψ is satisfied
```

Figure 6.1: Counterexample-based invariant refinement algorithm

each step in a set of global symbolic states $\mathcal{P}$, until the initial state *Init* is reached or until the preimage is empty.

To ensure termination, at each step, the visited symbolic state set $\mathcal{V}$ relative to the previous iterations is eliminated using the subtraction operator $\setminus$ in order to push the algorithm towards the initial state (line 10), else to conclude, if there is no intersection between $\mathcal{P}$ and *Init* and if $\mathcal{P}$ is empty, that (l^θ, ζ^θ) does not contain any valid counterexample. The set $\mathcal{V}$ is cumulative: it contains the states that have been visited during the analysis of the previous counterexamples. They are all eliminated during the subtraction operation. If there exists a symbolic state $s_0 \in \mathcal{P} \cap \mathcal{I}$, then the length of the shortest path from s_0 to (l^θ, ζ^θ) is equal to the number of preimage computation operations required to reach s_0. For each analyzed counterexample, we denote by the depth d the shortest path from (l^θ, ζ^θ) to the first backwards reachable state belonging to $\mathcal{I}$. If such a state does not exist, that is if the backward reachability algorithm reaches a set of symbolic states that has an empty preimage and has no intersection with $\mathcal{I}$, then the counterexample is spurious and the global invariant can be strengthened with its negation (line 14). We note that the operators $\setminus$ and $\cap$ on global symbolic state sets are slightly different from the usual set difference and conjunction operations on sets since symbolic states are defined by locations and zones. We consider the case where

the zone of a symbolic state from a given set is strictly included in the zone of a symbolic state from another set and has its same location.

Definition 6.1 (Predecessor of a symbolic state)**.** *Given $S = (L, \gamma, \mathcal{X}, T_\gamma, \mathsf{tpc}, \bar{s}_0)$ a timed system, we define the predecessor of a symbolic state $s' = (\bar{l}', \zeta')$ as the set*

$$pre(s') = \left\{ (\bar{l}, \zeta) \mid \exists \tau \in T_\gamma . \tau = (\bar{l}, \alpha, g, r, \bar{l}') \wedge \left(\zeta =_{\swarrow} \left(g \wedge [r](\zeta' \wedge \bigwedge_{x \in r} (x = 0)) \right) \right) \right\}.$$

For simplicity, we use the same notation to refer to the predecessor of a set of states $\mathcal{O}$, which is equal to the union of predecessor sets of its elements: $pre(\mathcal{O}) = \cup_{s \in \mathcal{O}} pre(s)$.

Given an interaction $\alpha \in \gamma$, we denote by $pre_\alpha(s')$ the set containing the predecessors of s' through interaction α.

6.2 Manipulation of Sets of Symbolic States

We note that a symbolic state is included in a set of symbolic states not only if it is equal to one of its included symbolic states, but even when its zone is a strict subset of a zone of a state in this set, given that their locations are identical. Besides, it is included in a set if its zone is included in the union of zones of a subset of symbolic states sharing its location. Thinking of a symbolic state (l, ζ) as a set of symbolic states whose zones are included in ζ, we formalize the *Join* and *Discard* operators as follows.

Definition 6.2 (Join Operator)**.** *Given two sets of symbolic states $\mathcal{O}$ and $\mathcal{O}'$, we define the set $\mathcal{O} \cap \mathcal{O}'$ as*

$$\mathcal{O} \cap \mathcal{O}' = \{ (l, \zeta \cap \zeta') \mid (l, \zeta) \in \mathcal{O} \wedge (l, \zeta') \in \mathcal{O}' \}$$

where $l = (l_1, \ldots, l_n)$ is an n-tuple of locations l_i of different components B_i.

Intuitively, discarding a symbolic state from a symbolic state set requires that its zone or any part of it should be subtracted from the zone of each symbolic state sharing its location from the set.

Definition 6.3 (Discard Operator)**.** *Discarding a symbolic state set $\mathcal{O}'$ from $\mathcal{O}$ results in the following set.*

$$\{ (l, \zeta) \in \mathcal{O} \mid l \notin Locations(\mathcal{O}') \} \cup \left\{ (l, \zeta \setminus \bigcup_{(l, \zeta') \in \mathcal{O}'} \zeta') \mid (l, \zeta) \in \mathcal{O}) \right\}$$

Example 6.1.
We consider the system composed of 3 components B_1, B_2 and B_3 having for clocks x, y and z respectively. Let

$$\mathcal{O} = \{((l_1^1, l_1^2, l_2^3), x \geq 5 \wedge y \leq 10 \wedge z < 3), ((l_2^1, l_1^2, l_2^3), x \leq 5 \wedge y \geq 2 \wedge z < 10)\}$$

where l_1^i denotes a location of B_i component.
Let the set $\mathcal{O}'$ be defined as

$$\mathcal{O}' = \{((l_1^1, l_1^2, l_2^3), x \geq 7 \wedge y \leq 8 \wedge z < 3))\}$$

let the set $\mathcal{O}''$ be defined as

$$\mathcal{O}'' = \{((l_1^1, l_1^2, l_2^3), 4 \leq y \leq 6), ((l_2^1, l_1^2, l_1^3), x \geq 5 \wedge y \leq 10 \wedge z < 3)\}$$

and let the set $\mathcal{O}^3$ be defined as

$$\mathcal{O}^3 = \{((l_1^1, l_1^2, l_2^3), x \geq 3 \wedge y \leq 12 \wedge z < 3)\}$$

- Discarding all the configurations of $\mathcal{O}'$ from $\mathcal{O}$ results in the following set

$$\{((l_1^1, l_1^2, l_2^3), 5 \leq x < 7 \wedge y \leq 10), ((l_1^1, l_1^2, l_2^3), x \geq 7 \wedge 8 < y \leq 10 \wedge z < 3) \\ , ((l_2^1, l_1^2, l_2^3), x \leq 5 \wedge y \geq 2 \wedge z < 10)\}$$

- Discarding all the configurations of $\mathcal{O}''$ from $\mathcal{O}$ results in the following set

$$\{((l_1^1, l_1^2, l_2^3), x \geq 5 \wedge 6 < y \leq 10 \wedge z < 3) \\ , ((l_1^1, l_1^2, l_2^3), x \geq 5 \wedge 0 \geq y < 4 \wedge z < 3) \\ , ((l_2^1, l_1^2, l_2^3), x \leq 5 \wedge y \geq 2 \wedge z < 10))\}$$

- Discarding all the configurations of $\mathcal{O}^3$ from $\mathcal{O}$ results in the following set

$$\{((l_2^1, l_1^2, l_2^3), x \leq 5 \wedge y \geq 2 \wedge z < 10)\}$$

Limitation The subtraction operation on DBM is very costly in the sense that it generally engenders a non convex zone where the splitting is necessary. This problem was detailed in [DHLP04] where an algorithm was proposed in order to perform more efficiently the DBM subtraction and to ensure that the generated DBMs are disjoint, for redundancy avoidance and to make the number of splits minimal.

In this work, we choose to work rather in a simpler form of symbolic set subtraction. A symbolic state (l, ζ) from $\mathcal{O}$ *belongs* henceforth to a set $\mathcal{O}'$ to discard only if there exists a state (l, ζ') in $\mathcal{O}'$ such that $\zeta \subset \zeta'$. This would ignore to subtract zones ζ which are partly included in a given ζ' from $\mathcal{O}'$ and would allow in some way the redundant and partial analysis of a previous symbolic state. Nevertheless, this approach is selected at the aim of drastically reducing the cost of the *set difference* operation:

Definition 6.4 (Set difference Operator)**.** *Given two sets of symbolic states $\mathcal{O}$ and $\mathcal{O}'$, we define the set $\mathcal{O} \setminus \mathcal{O}'$ as*

$$\mathcal{O} \setminus \mathcal{O}' \quad = \{(l, \zeta) \in \mathcal{O} \,|\forall (l', \zeta') \in \mathcal{O}'. l' \neq l \vee \zeta \not\subset \zeta'\}$$

Example 6.2. By using the set difference operator and based on the sets predefined in Example 6.1, $\mathcal{O} \setminus \mathcal{O}' = \mathcal{O}$ and $\mathcal{O} \setminus \mathcal{O}'' = \mathcal{O}$ while $\mathcal{O} \setminus \mathcal{O}^3 = \{((l_{21}, l_{12}, l_{23}), x \leq 5 \wedge y \geq 2 \wedge z < 10)\}$

This definition is adopted along the algorithm in Figure 6.1, at line 8 precisely. It is clear that it yields a more compact result than the previously defined discard operator.

6.3 Generalization of the counterexamples

In the timed case, the SAT-solver generates well-defined locations of components together with a precise valuation ν of the clock variables of the counterexample (line 3). The clock valuation ν is the conjunction of equalities of the form $x_{ij} = c_{ij}$ where the variable x_{ij} is a clock of the component B_i^h and c_{ij} is the constant which it equals in the solution generated by the SAT-solver.

If the refinement is done with respect to these clocks values, infinite number of counterexamples would be generally generated. As the clocks space is infinite ($\mathbb{R}_{\geq 0}$), and in order to ensure the termination of the iteration in the satisfiability checking loop, each counterexample needs to be *generalized*, before backward analysis, to a global symbolic state aggregating a set of counterexample states having the same location and whose clock valuations satisfy the same constraints. The choice of the generalization zone raises the trade-off between the complexity of the backward analysis of a given counterexample and the complexity of the space partitioning induced by the generalization of all the counterexamples.

The zone of the generated counterexample is generalized as follows:

$$generalize(\theta, \Psi) = (l, \bigwedge_{z_k \in \mathcal{L}.\nu \models z_k} z_k \wedge \bigwedge_{z_k \in \mathcal{L}.\nu \not\models z_k} \neg z_k)$$

where $\mathcal{L}$ stands for the set of literals constraining the clocks in the property Ψ. The generalization reflects which literals of the safety property are satisfied by the counterexample or not.

Example 6.3.

- We consider the property $\Psi \equiv (at(l_2^1) \wedge at(l_1^3) \Rightarrow x_1 \geq 5 \wedge x_2 \geq 8)$. We suppose that the counterexample θ generated by the SAT-solver is :

$$\begin{aligned} \theta = ((l_2^1, l_1^2, l_1^3), x_1 = 4 \wedge x_2 = 10 \wedge x_3 = 0 \wedge h_0 = 9 \\ \wedge h_{a_1} = 3 \wedge h_{a_2} = 1 \wedge h_{a_3} = 6 \\ \wedge h_{b_1} = 11 \wedge h_{b_2} = 8 \wedge h_{b_3} = 0) \end{aligned}$$

  where l_2^1, l_1^2 and l_1^3 are locations, , x_1 x_2 and x_3 are the clocks of components B_1, B_2 and B_3 respectively and h_{a_1}, h_{a_2}, h_{a_3} h_{b_1}, h_{b_2} and h_{b_3} are history clocks. The generalized counterexample is:

$$generalize(\theta, \Psi) = ((l_2^1, l_1^2, l_1^3), x_1 < 5 \wedge x_2 \geq 8)$$

Example 6.4.

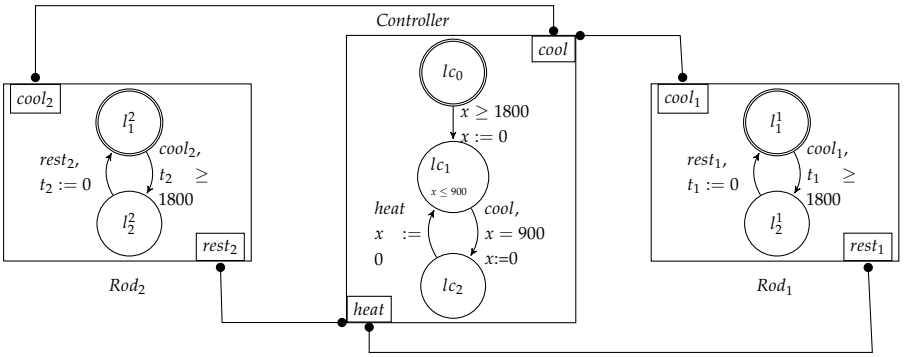


Figure 6.2: A timed temperature control system

For illustration, we consider the temperature control system depicted in Figure 6.2 from head to tail. It is composed of one controller interacting with two rods through interaction set $\gamma = \{heat|rest_1, heat|rest_2, cool|cool_1, cool|cool_2\}$.

- We consider the property $\Psi \equiv at((lc_1, l_2^1, l_1^2)) \Rightarrow (t_1 \geq 1800 \wedge t_1 - t_2 < 1500)$, which is satisfied. The SAT-solver checks if there is a configuration satisfying $GI \wedge \neg\Psi$. Using the tautology $(A \Rightarrow B \equiv B \vee \neg A)$, the negation of the safety property is:

$$\neg\Psi \equiv at((lc_1, l_2^1, l_1^2)) \wedge (t_1 < 1800 \vee t_1 - t_2 \geq 1500)$$

The counterexample generated by the SAT-solver θ has for location (lc_1, l_2^1, l_1^2) and has the following clock valuations:

location	x	t_1	t_2	h_0	h_{cool}	h_{cool_1}	h_{cool_2}	h_{heat}	h_{rest_1}	h_{rest_2}
(lc_1, l_2^1, l_1^2)	451	2702	451	4502	901	901	2251	451	2702	451

Table 6.1: The generated counterexample

The generalized counterexample is expressed as

$$generalize(\theta, \Psi) = ((lc_1, l_2^1, l_1^2), t_1 \geq 1800 \wedge t_1 - t_2 \geq 1500)$$

The state $generalize(\theta, \Psi)$ satisfies the literal $(t_1 - t_2 \geq 1500)$ of $\neg\Psi$ and negates its $(t_1 < 1800)$ literal. The computed predecessors of $generalize(\theta, \Psi)$ are listed in the following. We refer with $\mathcal{O}_i$ to the set $\mathcal{O}$ obtained at the i-th iteration, that is at depth i. We relate also the number of symbolic states discarded from the preimage at each step. We recall that $\mathcal{O}_i \leftarrow pre(\mathcal{O}_{i-1}) \setminus \mathcal{V}_i$.

The backward reachability graph that follows from the algorithm is depicted in Figure 6.3. It shows the computed preimage at each depth i of backward reachability.
For example, for $i = 0$, $\mathcal{O}_0 = \{((lc_1, l_2^1, l_1^2), z_0)\}$, and for $i = 1$, $\mathcal{O}_1 = pre(((lc_1, l_2^1, l_1^2), z_0)) = \{((lc_2, l_2^1, l_2^2), z_1)\}$.
The graph indicates whether a reached symbolic state is discarded from the preimage using the set difference operation in line 8 of the algorithm:

- If the predecessor of a symbolic state $(\bar{l}, \zeta)$ equals a previously analyzed symbolic state then the red arrow is used to return to the firstly computed state. For our example:

$$pre(((lc_2, l_1^1, l_2^2), z_5)) = pre_{cool|cool_1}(((lc_2, l_1^1, l_2^2), z_5)) \cup pre_{cool|cool_2}(((lc_2, l_2^1, l_1^2), z_5))$$

where

$$pre_{cool|cool_1}(((lc_2, l_1^1, l_2^2), z_5)) = \{(lc_1, l_1^1, l_2^2), z_{44})\}$$

$$pre_{cool|cool_2}(((lc_2, l_2^1, l_1^2), z_5)) = \{(lc_1, l_1^1, l_2^2), z_{43})\}$$

- If the zone ζ of a symbolic state (l, ζ) in $pre(\mathcal{O}_{i-1})$ is included in the zone ζ' of a previously analyzed symbolic state (l, ζ') sharing its location, then it is discarded from $\mathcal{O}_i$. Such states are highlighted with the green color. For our example, $z_{41} \subset z_{21}$, $z_{42} \subset z_{43}$, and $z_{44} \subset z_{21}$.
- The symbolic states having no predecessors are underlined with the red cross mark.

6.4 Incremental Restriction of Pre-image Computation

To avoid combinatorial explosion during the exploration of the predecessors, guiding the backward analysis by the global invariant GI at each step would reduce the overall number of analyzed states. Guiding the analysis with GI means that each backward computed state is restricted to satisfy it.This restriction is denoted by pre^{GI} and defined by $pre^{GI}(\mathcal{O}) = \{s \in pre(\mathcal{O}) \mid s \models GI\}$. If This restriction is denoted by pre^{GI} and defined by $pre^{GI}(\mathcal{O}) = \{s \in pre(\mathcal{O}) \mid s \models GI\}$.

The invariant GI is computed as the conjunction of local invariants of components and invariants constraining the history clocks relations. Transforming the GI predicate into a CNF or DNF form is very costly and complex, especially when the number of components is great. This leads us to restrict in an incremental way each state in a given set to satisfy GI, hence the different invariants composing GI should be satisfied.

6.4.1 Restriction with $II(\gamma)$

As the interaction invariant constraints only locations, we consider to firstly refine each state with respect to it. This allows to eliminate some global symbolic states whose locations don't satisfy II, preventing us from looking at their zones. The symbolic states in a given set $\mathcal{O}$ which satisfy $II(\gamma)$ are included in:

$$refine_{II}(\mathcal{O}) = \{(l, \zeta) \in \mathcal{O} \mid l \models II(\gamma)\}$$

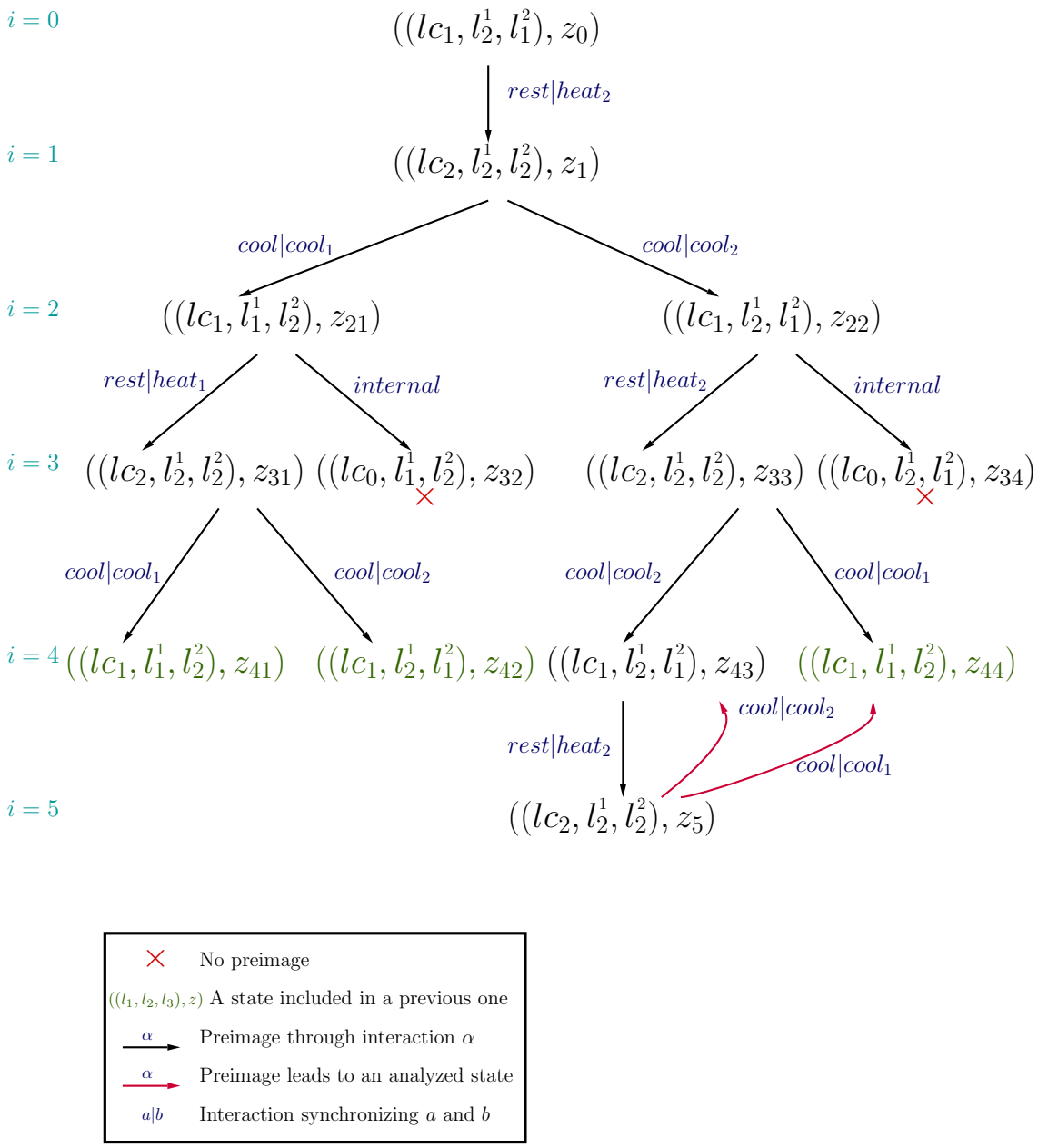


Figure 6.3: Graph resulting from the counterexample analysis algorithm

6.4.2 Restriction with $CI(B_i^h)$

After computing $\mathcal{O}^{(0)} = refine_{II}(\mathcal{O})$, we define the restriction of a set $\mathcal{O}^{(i-1)}$ with $CI(B_i^h)$, the local invariant of component B_i:

$$restrict_{CI}(\mathcal{O}^{(i-1)}, CI(B_i^h)) = \left\{(l, \zeta \cap \zeta^{ij}) \mid (l, \zeta) \in \mathcal{O}^{(i-1)} \wedge (l_i, \zeta^{ij}) \in Reach(B_i^h)\right\}$$

Given that components are numbered $B_1, \cdots B_n$, the set resulting from the restriction with components invariants is computed as follows

For $i = 1 \ldots n$

$$\mathcal{O}^{(i)} = restrict_{CI}(\mathcal{O}^{(i-1)}, CI(B_i^h))$$

6.4.3 Restriction with the history clocks constraints

Thereafter, the state set is refined with the history clocks constraints $\mathcal{E}^*(\gamma)$ and $\mathcal{S}(\gamma)$. Both of them can be perceived as a disjunction of zones, or equivalently sets of zones. The final set of restricted states is

$$restrict(\mathcal{O}, GI) = \left\{(l, \zeta \cap \zeta_{\mathcal{E}^*} \cap \zeta_{\mathcal{S}}) \mid (l, \zeta) \in \mathcal{O}^{(n)} \wedge \zeta_{\mathcal{E}^*} \in \mathcal{E}^*(\gamma) \wedge \zeta_{\mathcal{S}} \in \mathcal{S}(\gamma)\right\}$$

Every state in $restrict(\mathcal{O}, GI)$ satisfies GI. Concretely, the restriction of $pre(\mathcal{O})$, the set $pre^{GI}(\mathcal{O})$, is

$$pre^{GI}(\mathcal{O}) = restrict(pre(\mathcal{O}), GI)$$

Example 6.5.
We reconsider Example 6.4 and apply the restriction operation to the analyzed symbolic states. The new graph is shown in Figure 6.4. Before computing its preimage, the generalized counterexample is restricted with respect to GI. This results in two symbolic states sharing the same global location and differing in their zones. Afterwards, the preimage of each of them is computed. The preimage contains 4 different symbolic states. However, they are all eliminated since they do not satisfy the global invariant. All in all, 7 symbolic states are computed, instead of 13 when no restriction with GI is performed. In addition, the elimination of this spurious counterexample is performed in fewer backward computation steps.

6.4.4 Discussion

In general, the restriction with the global invariant allows to reduce the number of backward computation steps. In fact, the preimage of the sub-states which

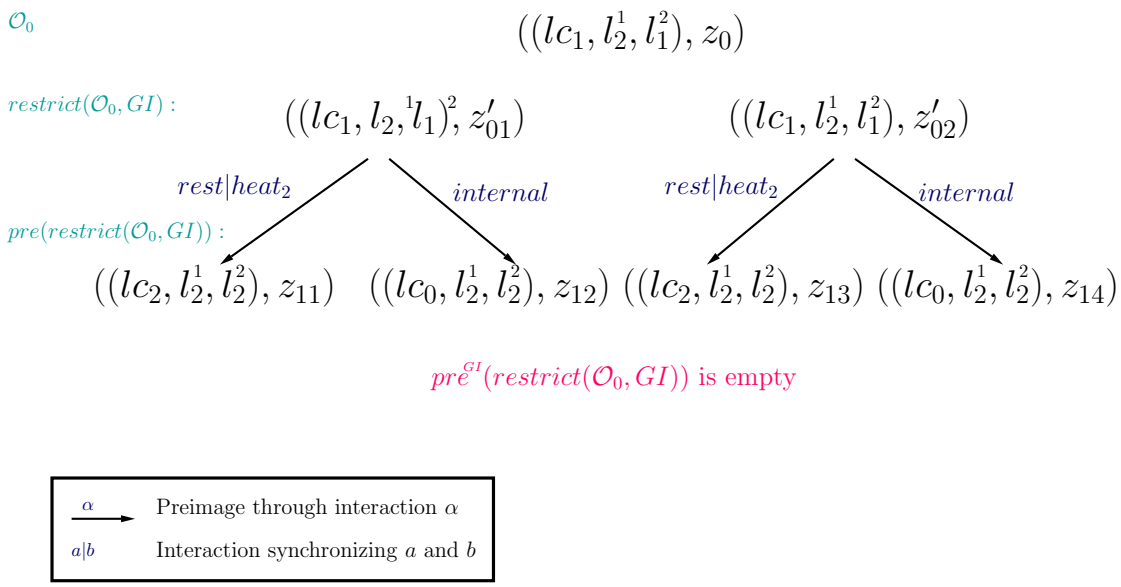


Figure 6.4: Graph resulting from the counterexample analysis algorithm applying restriction with GI

do not satisfy the global invariant is not performed, helping to reduce the size of the overall explored state space. The restriction with the global invariant drastically reduced the overall verification time for the untimed systems introduced in [BGL+11]. However, the use of BDD is not possible for timed systems with the existence of the real variables representing the clocks.
The restriction with the global invariant in case of timed systems with large number of components needs to be efficiently implemented. In fact, the number of zones representing the history clock constraints increases significantly with the number of interactions. In some way, a partitioning of each backward reached symbolic state with respect to the zones of the global invariant is required. Therefore, the use of efficient data structures is necessary. Furthermore, in order to reduce the complexity of the restriction operation, some previous techniques already proposed for monolithic verification would be of great use. We cite for example symmetry reduction techniques. Such techniques may be also applicable for the basic algorithm where the preimage computation is not guided.

6.5 Summary

While the global invariant computation gives a first attempt to check the property by avoiding the full state space exploration, this chapter proposes the analysis of

the counterexamples as a second stage to ensure the completeness of the method. The validation of a counterexample requires to find a backward path showing that the suspected configuration can be reached from the initial state. Guiding the basic backward computation with the global computed invariant would allow the minimization of the number of the preimage computation operations. However, regarding the particular form of the invariant, the restriction of the reached preimages requires the use of efficient data structures.

The application of reduction techniques for the backward computation module remains open to investigation. The implementation of the previous methods already proposed to alleviate the complexity of monolithic verification to our backward reachability analysis module would ensure that the verification time and complexity of our method remains reduced in comparison with other techniques. In case a valid counterexample exists, the algorithm could be extended to extract the error trace.

Chapter 7

Verification of Parameterized Timed Systems

Contents

The parameterized verification proposes to check whether a uniform family of systems satisfies a desired property. By uniform family, we refer to systems that differ solely on the number of copies of a same replicated component. As examples of parameterized systems, we cite communication protocols, traffic control protocols and swarm robots. The verification of such systems is challenging by that they yield infinite-state space since the system description is parameterized by a number of components while the verification aims to establish the correctness independently from their number. Following from the *Halting Problem* for Turing Machines, the parameterized model-checking problem is generally undecidable. The proof is based on that a system of a given size n can simulate a Turing machine

with the same number n of steps [AK86].
To remedy this problem, some research focused on restricted classes of parameterized systems where the problem is decidable. For instance, decidability is insured for the parameterized verification of safety properties for timed networks where each timed automata possesses either a finite number of discrete-valued clocks or a single real-valued clock [CGR10, AJ03]. In [ADM05], special classes of timed networks were studied. In case of timed networks containing exclusively non-strict inequalities on the clock variables, the model-checking problem is decidable independently from the number of clocks in each component. At the opposite, in case of *open* timed networks where inequalities on clocks are all strict, the undecidablilty remains.
One other direction of research accepts to give up on completeness for applicability to more general classes of parameterized timed systems by relying on approximation methods. We mention regular model-checking [BJNT00, JN00, AJNd02], abstraction and network invariants ([WL90, AJ99, LHR01] for untimed systems and [GL08] for timed systems).
In [CGR10], Satisfiability Modulo Theories (SMT) techniques are used to translate timed and parameterized reachability problems to the declarative input language of MCMT, the Model Checker Modulo Theories tool. For this purpose, the components are encoded as formula in a decidable fragment of the arrays theory [GNRZ08].
In the remainder of this chapter, an extension of our compositional verification method is detailed. It is built upon a *small model theorem* proposed for *rectangular* hybrid automata [JM12]. In [PRZ01], it has been demonstrated that for *bounded-data parameterized systems* and for a precise category of properties, there exists always a number n called the *cutoff* such that if the property is satisfied for any number of components smaller than n, then it is also guaranteed for any number of components beyond n. The work in [JM12] aims to find a method for the computation of such a number in the context of hybrid rectangular automata. Our compositional verification method offers an elegant application of the results in [JM12] and shows relatively small formulas which express the computed invariant in a parameterized setting. This chapter first relates the prerequisite results, then recalls the semantics of parametrized timed systems and finally shows the adaptation of our compositional verification to fit into this framework.

7.1 A Small Model Theorem

In [JM12], the authors introduced a class of assertions called *LH-assertions* and described how they can be used to express inductive invariants of rectangular hybrid automata (RHA) systems, that is systems combining finite state machines with continuous variables [HKPV98, ACH+95]. Networks of timed automata can be perceived as a subclass of RHA systems and T-assertions are introduced as a particular case of LH-assertions. They are used to assert on local and system properties in the framework of this subclass of systems.

T-assertions

A and n are natural numbers in $\mathbb{N}$. The signature of a T-assertion involves a finite number of variables of different types:

(a) *index variables*: $i_1, \ldots, i_a \in \mathbb{N}$

(b) *discrete variables*: $l_1, \ldots, l_b \in L$

(c) *real variables*: $x_1, \ldots, x_c \in \mathbb{R}$

(d) *discrete array variables*: $\bar{l}_1, \ldots, \bar{l}_d : [n] \to L$

(e) *real array variables*: $\bar{x}_1, \ldots, \bar{x}_e : [n] \to \mathbb{R}^+$

where $[n]$ denotes the set $\{1, \ldots, n\}$.
The terms of a T-assertion respect the following BNF grammar:

$$\texttt{ITerm} ::= 1 \mid n \mid i_j \qquad \texttt{DTerm} ::= L_j \mid l_k \mid \bar{l}_j[\texttt{ITerm}] \qquad \texttt{RTerm} ::= x_j \mid \bar{x}_k[\texttt{ITerm}]$$

An index term `ITerm` can be one of the constants 1 or N or an index variable i_j. The discrete terms in `DTerm` are constructed as above specified, where L_j is a constant in L, l_k is a discrete variable and $\bar{l}_j$ is a discrete array. The definition of the real terms in `RTerm` is specified by use of real variable x_j and a real array $\bar{x}_k$. The formulas are defined structurally as:

$$\texttt{Atom} ::= \texttt{ITerm} < \texttt{ITerm} \mid \texttt{DTerm} = L_k \mid a \cdot \texttt{RTerm} + b \cdot \texttt{RTerm} + c < 0$$
$$\texttt{Formula} ::= \texttt{Atom} \mid \neg\texttt{Formula} \mid \texttt{Formula} \wedge \texttt{Formula}$$

where a, b and c are real-valued variables.
A formula `Formula` can be obtained by joining shorter formulas using the boolean conjunction operator. The boolean disjunction operator $\vee$ and other comparison operators like $=$, $\neq$, $\leq$ and $\geq$ can be derived by combining $\wedge$ with $\neg$ and $<$. For example $i \neq j$ is equivalent to $\neg(\neg(i < j) \wedge \neg(j < i))$. A formula with no free

variables can be obtained from a `Formula` F by quantification and is called a *sentence*.

A T-assertion is a sentence of the following form:

$$\forall i_1, \ldots, i_k \in [n]\ \exists j_1, \ldots, j_m \in [n].F$$

In the context of parameterized timed systems, the elements of the above grammar are used as follows: the different index variables denote the different instances of the replicated component, the discrete array variables $\bar{l}_1, \ldots, \bar{l}_d$ denote the locations and the real array variables $\bar{x}_1, \ldots, \bar{x}_e$ denote the clocks.
For brevity, in the rest of this chapter, $\overline{x}[i+o]$ is used to stand for $\exists j.j = i + o \wedge \overline{x}[j]$.

T-assertions are used for instance to express safety properties of timed systems.

Example 7.1 (T-assertions expressing safety properties)**.** We consider for illustration the following T-assertions:

1. There is a minimum time delay between the clocks of two different processes being at the same location:

$$\forall i, j.(\bar{l}[i] = \bar{l}[j] \rightarrow \mid \bar{x}[i] - \bar{x}[j] \mid \geq 2)$$

2. Two different processes cannot be in the critical state CS at the same time:

$$\forall i, j.(i = j) \vee \neg(\bar{l}[i] = CS \wedge \bar{l}[j] = CS)$$

Models for Assertions As in [JM12], a *n-model* gives interpretation to the elements for a given assertion. More precisely, it provides the ranges or the values to which the variables may be assigned. For an assertion ψ, the n-model is denoted by $M(n, \psi)$. A n-model satisfies an assertion ψ if ψ evaluates to true with the assignments of the free variables given by $M(n, \psi)$. For example, a 3-model for the minimum delay property is $\bar{l} = [L_1, L_1, L_2]$, $\bar{x} = [1, 3, 2]$. An assertion is said to be *valid* if all of its models satisfy it. If some models satisfy it and some don't, then it is *satisfiable*.
In [JM12], a small model theorem was proposed for LH-assertions. It shows that inductive invariants for networks of RHA can be expressed as LH-assertions. Besides, it provides a threshold on the size of models such that if for all n-models with n smaller than the threshold the LH-assertion is satisfied, then it is satisfied by all the n-models. In the following proposition, a restriction of the theorem to T-assertions is formulated.

Proposition 7.1 (Adapted from [JM12]). *Let Φ be a T-assertion in the form $\forall i_1, \ldots, i_k \in [n]\ \exists j_1, \ldots, j_m \in [n].\phi$ where ϕ is a quantifier-free formula involving the index variables $i_1, \ldots, i_k$, $j_1, \ldots, j_m$ and array variables.*
The assertion Φ is valid iff, for all $n \leq k + 2$, Φ is satisfied by all n-models.

In the following of this chapter, we show how the small model theorem can be applied to the global inductive invariant that is detailed in Chapter 5.

7.2 Syntax and Semantics of Parameterized Timed Systems

In our framework, a Parameterized Timed System (PTS) contains two families of components: optionally components whose number is fixed and whose composition can be perceived as a single non parameterized component C, and the isomorphic components B^i whose number is parameterized by n. All the components are modeled as in Chapter 4. The system resulting from the composition of the two parts is denoted $C \|_\gamma^n B_i$. We denote by A_c, A and A^i the set of actions of C, the generic component B and the component B^i respectively. The set A^i is obtained from A by attaching to each action the index i.

Example 7.2 (A Parameterized Timed System). Figure 7.1 depicts a parameterized timed system where a controller C interacts with a set of isomorphic components B^i through interaction set $\{a|a^i\}$. The components B^i are obtained from the generic component B by adding the index i to the actions, locations and clocks. During the execution of a transition labeled a^i together with a^c, the clocks x^i and x^c are reset simultaneously.

Interaction Patterns In the parameterized setting, the interaction set is redefined by interaction patterns instead of a fixed set of interactions. An interaction pattern allows to define at an abstract level a family of interactions between the component C and m components. It is defined by a tuple $\left(a^c, (a_1, o_1), \ldots, (a_m, o_m)\right) \in A^c \times (A \times \mathbb{N})^m$ where $0 = o_1 < o_2 < \cdots < o_m$. For example, the pattern $\left(a^c, (a_1, o_1), \ldots, (a_m, o_m)\right)$ generates n interactions $\alpha^i = \left(a^c, a_1^{i+o_1}, \ldots, a_m^{i+o_m}\right) \in A^c \times A^{i+o_1} \times \cdots \times A^{i+o_m}$ such that all the terms are though of modulo n. The notation $gen(\alpha)$ is used for the interactions generated by α. Depending from the context, γ denotes either the set of interaction patterns, or $\cup_{i \in [n]} \alpha^i$.

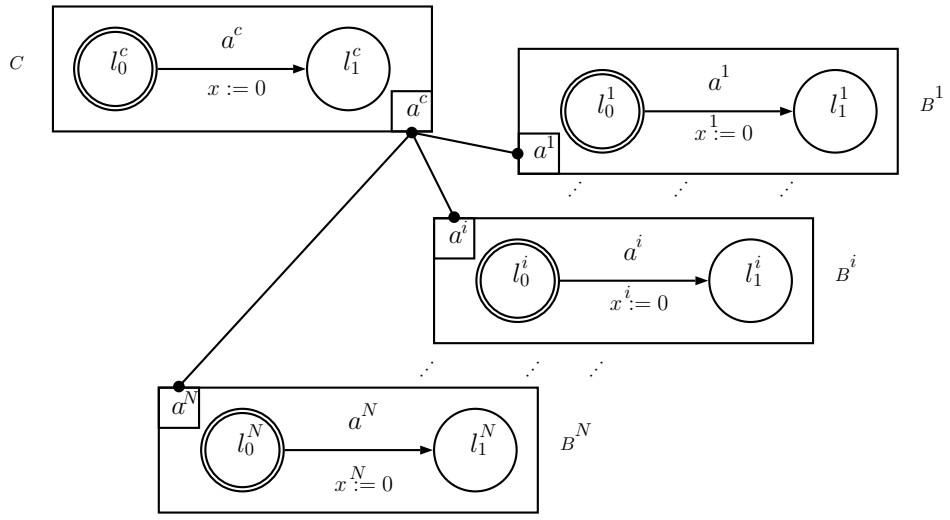


Figure 7.1: A parameterized timed system

Example 7.3. The interaction pattern for the parameterized system in Figure 7.1 is $\alpha = \big(a^c, (a, 0)\big)$ and results in the set of interactions $\{(a^c, a^1), (a^c, a^2), \ldots, (a^c, a^n)\}$.

Equivalently to timed systems shown in Chapter 4, the semantics of a parameterized timed system is defined by a labeled transition system as follows.

Definition 7.1. *Semantics of Parameterized Timed Systems*
Given $C = (L^c, A^c, T^c, \mathcal{X}^c, \mathsf{tpc}^c, s_0^c)$ *and* $B^i = (L^i, A^i, T^i, \mathcal{X}^i, \mathsf{tpc}^i, s_0^i)$*, the semantics of the system obtained from the parallel composition of* C *with* n *components* B^i *is equal to the semantics of the component* $C\|_\gamma^n B_i$*. It corresponds therefore to the semantics of the timed component* $(L, \gamma, \mathcal{X}, T_\gamma, \mathsf{tpc}, s_0)$*, where*

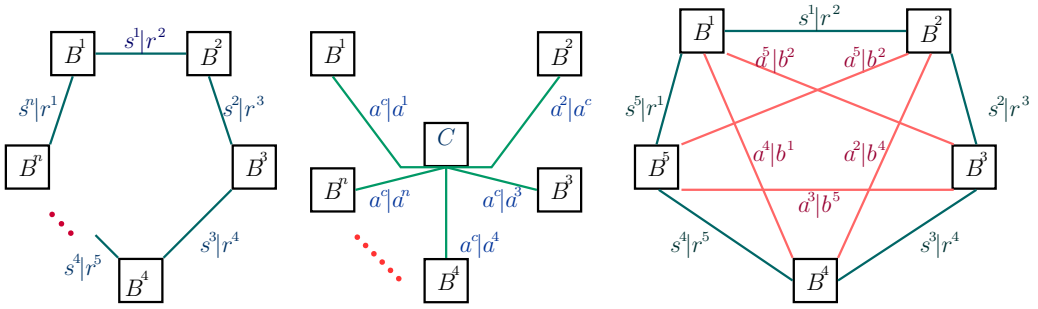
- $L = L^c \times_i L^i$ *is the set of global locations.*
- $s_0 = ((l_0^1, ..., l_0^n), \bigwedge_i c_0^i)$ *, where* $\forall i$*,* $s_0^i = (l_0^i, c_0^i)$*.*
- $\mathcal{X} = \mathcal{X}^c \cup_i \mathcal{X}^i$ *is the set of clocks.*
- $\mathsf{tpc}(\bar{l}) = \mathsf{tpc}(l^c) \bigwedge_i \mathsf{tpc}(l^i)$*,* $\forall \bar{l} = (l^c, l^1, ..., l^n) \in L$*.*
- *The set of global transitions is defined by*

$$T_\gamma = \left\{ (\bar{l}, (\alpha^i, g, r), \bar{l'}) \;\middle|\; \begin{array}{l} \bar{l} = (l^c, l^1, ..., l^n) \in L,\ \ \bar{l'} = (l'^c, l'^1, ..., l'^n) \in L \\ \alpha^i = \big(a^c, a_1^{i+o_1}, \ldots, a_m^{i+o_m}\big) \text{ and } \mathcal{O} = \{i + o_1, \ldots, i + o_m\} \\ \forall i \in \mathcal{O}.(l^i, (a^i, g^i, r^i), l'^i) \in T^i, a^i \in A_i \cap \alpha^i, \forall i \notin \mathcal{O}.l^i = l'^i \\ (l^c, (a^c, g^c, r^c), l'^c) \in T^c \\ r = r^c \bigcup_{i \in \mathcal{O}} r^i, g = g^c \bigwedge_{i \in \mathcal{O}} g^i \end{array} \right\}$$

Interaction Patterns as Topologies In the field of graph theory, a topology is the arrangement of different elements depicted logically by vertices and edges. Interaction patterns have a straightforward encoding as topology graphs where the vertices model the components and the edges model the communication between them.

Figure 7.2 illustrates star, bus, ring and mesh topologies. The star topology connects a controller C with a set of identical components whereas the star and mesh topologies contain purely identical components. The interaction set of the star topology is generated by the pattern $(a^c, (a, 0))$. The classical ring topology is generated by the pattern $((s, 0), (r, 1))$. The corresponding interaction set links each *send* action from a node i, abbreviated with s, to a *receive* action, abbreviated r, from the node $i + 1$.

The mesh topology depicted in Figure 7.2 illustrates an interaction set obtained by combination of the two patterns $((s, 0), (r, 1))$ and $((a, 0), (b, 2))$. It shows its instantiation to the number 5 of components.



(a) Ring topology (b) Star topology (c) Mesh topology

Figure 7.2: Topologies for interaction structures

7.3 Formulating Invariants for Parameterized Timed Systems

In this section, we show how the global invariant detailed in Chapter 5 can be perceived as a T-assertion in the context of parameterized timed systems. This allows for the application of the small model theorem with the compositional verification method.

We recall that the verification rule proposes to prove, for a fixed number n of

components, the validity of the following formula:

$$CI(C) \wedge \bigwedge_i CI(B_i) \wedge II(\gamma) \wedge \mathcal{E}(\gamma) \wedge \mathcal{S}(\gamma) \Rightarrow \Psi$$

This reduces to the unsatisfiability checking of $GI \wedge \neg\Psi$. In the following, we show how the invariants composing GI can fit in the class of T-assertions.

7.3.1 Component Invariant

As related in Chapter 5, this invariant is computed as the set of the reachable states of the component and is expressed as the disjunction of predicates $at(l_j) \wedge \zeta_i$. Each of them is valid when the component is at the reachable state (l_i, ζ_i).
If the generic component has an action a, a clock x and a location l, then the corresponding notations in the component B^i are referred to by a^i, x^i and l^i respectively, In order to shape the component invariants into T-assertions, the clock x^i is viewed as the ith element of the array $\bar{x}$ and the location l^i is perceived as the ith element of the array $\bar{l}$. Semantically, $\bar{l}[i]$ and l^i are equivalent, as well as $\bar{x}[i]$ and x^i.

Example 7.4. The component invariants of C, B and B^i (where C, B^i, B are the ones depicted in Figures 7.1) are:

$$CI(C) \equiv (l_0^c \wedge x^c \geq 0) \vee (l_1^c \wedge x^c \geq 0) \tag{7.1}$$

$$CI(B) \equiv (l_0 \wedge x \geq 0) \quad \vee \quad (l_1 \wedge x \geq 0)$$

$$CI(B^i) \equiv (\overline{l_0}[i] \wedge \overline{x}[i] \geq 0) \vee (\overline{l_1}[i] \wedge \overline{x}[i] \geq 0) \tag{7.2}$$

If we use the notation CI to denote the conjunction of the invariant of the component C, if exists, with the local invariants of all the replicated components B^i, then CI is an invariant of the parallel composition, following from that the conjunction of invariants is an invariant. Formally, CI is defined as $CI \stackrel{\triangle}{\equiv} \big(CI(C) \wedge \forall i.CI(B^i)\big)$.

7.3.2 History Clocks Constraints

As explained in Chapter 5, the history clocks aim to extract in the global invariant relations resulting from the time synchronization between components. To each action, an action history clock is introduced and is reset whenever the action occurs. Similarly, interaction history clocks are related to interactions. Given that interactions synchronize actions from the different components, all the related

clocks are reset at the same time. They remain equal until a more recent interaction resets one of them, making it smaller than all the others. In this section, the reformulation of the history clocks constraints $\mathcal{S}(\gamma)$ and $\mathcal{E}(\gamma)$ to fit in the parameterized framework is detailed. The equivalent invariants are denoted by $\bar{\mathcal{S}}(\gamma)$ and $\bar{\mathcal{E}}(\gamma)$ respectively.
To each interaction pattern α and each $a \in \alpha$, the array $\bar{h_\alpha}$, respectively $\bar{h_a}$ is associated.

History Clocks Inequalities Given α of the form $(\dots,(a,o),\dots)$, that is to say, for an index i, a^i appears in α^{i-o}. It results that $\overline{h_a}[i]$ is the minimum among $\overline{h_\alpha}[i-o]$.

$$\bar{\mathcal{E}}(a^i) \equiv \left(\bigvee_{(a,o)\in\alpha} \overline{h_a}[i] = \overline{h_\alpha}[i-o] \right) \wedge \left(\bigwedge_{(a,o)\in\alpha} \overline{h_a}[i] \leq \overline{h_\alpha}[i-o] \right)$$

Considering an action a^c belonging to the component C, the related inequality invariant is expressed as follows:

$$\bar{\mathcal{E}}(a^c) \equiv \exists j. \left(\bigvee_{a^c\in\alpha} h_{a^c} = \overline{h_\alpha}[j] \right) \wedge \forall i. \left(\bigwedge_{a^c\in\alpha} h_{a^c} \leq \overline{h_\alpha}[i] \right).$$

The existential quantifier serves to relate the unbounded number of interactions to which a^c belongs. Its history clock h_{a^c} is equal to the minimum among the history clocks of these interactions. The history clocks inequalities considering all the actions are gathered in $\bar{\mathcal{E}}(\gamma)$:

$$\bar{\mathcal{E}}(\gamma) \equiv \forall i \wedge_a \bar{\mathcal{E}}(a^i) \wedge_{a^c} \bar{\mathcal{E}}(a^c)$$

We note that this predicate is the equivalent to $\mathcal{E}^*(\gamma)$, detailed in Chapter 5, in the non parameterized case. The predicate $\bar{\mathcal{E}}(\gamma)$ is an invariant of $C^h \|_\gamma^n B^{ih}$.

Example 7.5.

1. For the star topology $\gamma = \{\alpha\}$ with $\alpha = \big(a^c, (a,0)\big)$ we have that:

$$\begin{aligned}\bar{\mathcal{E}}(\gamma) \equiv \forall i.(\overline{h_a}[i] = \overline{h_\alpha}[i] \wedge \overline{h_a}[i] \leq \overline{h_\alpha}[i]) \wedge \\ \exists j.(h_{a^c} = \overline{h_\alpha}[j]) \wedge \forall i.(h_{a^c} \leq \overline{h_\alpha}[i])\end{aligned}$$

2. As for the ring topology $\gamma = \{\alpha\}$ with $\alpha = \big((s,0),(r,1)\big)$ we have:

$$\begin{aligned}\bar{\mathcal{E}}(\gamma) \equiv \forall i.\big(&\overline{h_s}[i] = \overline{h_\alpha}[i] \wedge \overline{h_s}[i] \leq \overline{h_\alpha}[i]\big)\\ &\wedge \big(\overline{h_r}[i] = \overline{h_\alpha}[i-1] \wedge \overline{h_r}[i] \leq \overline{h_\alpha}[i-1]\big).\end{aligned}$$

3. (a) We consider the mesh topology depicted in Figure. 7.2 generated from the combination of the interaction patterns $\alpha = ((s,0),(r,1))$ and $\beta = ((a,0),(b,2))$. For the generic component the actions s, r, a and b are all distinct.

$$\begin{aligned}\bar{\mathcal{E}}(\gamma) \equiv \forall i.\big(&\overline{h_s}[i] = \overline{h_\alpha}[i] \wedge \overline{h_s}[i] \leq \overline{h_\alpha}[i]\big)\\ &\wedge \big(\overline{h_r}[i] = \overline{h_\alpha}[i-1] \wedge \overline{h_r}[i] \leq \overline{h_\alpha}[i-1]\big)\\ &\wedge \big(\overline{h_a}[i] = \overline{h_\beta}[i] \wedge \overline{h_a}[i] \leq \overline{h_\beta}[i]\big)\\ &\wedge \big(\overline{h_b}[i] = \overline{h_\beta}[i-2] \wedge \overline{h_r}[i] \leq \overline{h_\beta}[i-2]\big).\end{aligned}$$

   (b) To illustrate an interaction pattern with more conflicting interactions, we consider the mesh topology where the two combined patterns have common actions. We suppose that $\alpha = ((s,0),(r,1))$ and $\beta = ((s,0),(r,2))$

$$\begin{aligned}\bar{\mathcal{E}}(\gamma) \equiv \forall i.\big(&(\overline{h_s}[i] = \overline{h_\alpha}[i] \wedge \overline{h_\alpha}[i] \leq \overline{h_\beta}[i])\\ &\vee (\overline{h_s}[i] = \overline{h_\beta}[i] \wedge \overline{h_\beta}[i] \leq \overline{h_\alpha}[i]))\\ &\wedge \big((\overline{h_r}[i] = \overline{h_\alpha}[i-1] \wedge \overline{h_\alpha}[i-1] \leq \overline{h_\beta}[i-2])\\ &\vee (\overline{h_r}[i] = \overline{h_\beta}[i-2] \wedge \overline{h_\beta}[i-2] \leq \overline{h_\alpha}[i-1])\big).\end{aligned}$$

Separation Constraints In Section 5.2, the separation constraints predicate is defined as an invariant serving to tighten the over-approximation in case of conflicting interactions. In the following, we show how it can be reformulated in case of parameterized timed system to fit into the T-assertion grammar. We distinguish two types of actions, actions in the B^i components and actions in the component C.

$$\bar{\mathcal{S}}(a^i) \equiv \bigwedge_{\substack{(a,o_1)\in\alpha\\(a,o_2)\in\beta\\o_1\neq o_2 \vee \alpha\neq\beta}} \Big|\overline{h_\alpha}[i-o_1] - \overline{h_\beta}[i-o_2]\Big| \geq k_a$$

$$\bar{\mathcal{S}}(a^c) \equiv \forall i_1, i_2.\ \bigwedge_{\substack{\alpha\ni a^c\\\beta\ni a^c\\i_1\neq i_2 \vee \alpha\neq\beta}} \Big|\overline{h_\alpha}[i_1] - \overline{h_\beta}[i_2]\Big| \geq k_{a^c}$$

where the constant k_a (resp. k_{a^c}) is the minimum time elapse required between two occurrences of a (resp. a^c) action. Their computation is detailed in Section 5.2. The separation constraints invariant is formulated for PTS as

$$\bar{\mathcal{S}}(\gamma) \equiv \forall i \wedge_a \bar{\mathcal{S}}(a^i) \wedge_{a^c} \bar{\mathcal{S}}(a^c)$$

.

Example 7.6. For the star topology $\gamma = \{\alpha\}$ with $\alpha = \big(a^c, (a, 0)\big)$, we have that:

$$\bar{\mathcal{S}}(\gamma) \equiv \forall i_1, i_2. \left|\overline{h_\alpha}[i_1] - \overline{h_\alpha}[i_2]\right| \geq k_{a^c} \tag{7.3}$$

As the ring topology $\gamma = \{\alpha\}$, with $\alpha = \big((s, 0)(r, 1)\big)$, shows no conflicts, we consider for illustration purpose the following slight variation $\alpha = \big((r, 0), (s, 1), (r, 2)\big)$ corresponding to send actions being translated to the left and to the right. In this case, we have that:

$$\bar{\mathcal{S}}(\gamma) \equiv \forall i. \left|\overline{h_\alpha}[i] - \overline{h_\alpha}[i - 2]\right| \geq k_r \tag{7.4}$$

where k_r is the lower bound of the time elapsed between two consecutive r actions.

7.3.3 Interaction Invariant

Interaction invariant constrains the global locations of the system. It is deduced from the synchronization between the actions of the components based on the interaction structure and in total independence from all timing aspects. As explained in Chapter 3, the interaction invariant can be computed by boolean behavioral constraints or by resolving linear constraints. Applying both methods in the parameterized case is unpractical as it requires either the transformation of quantified formula into the disjunctive form or the resolution of an unbounded number of equations. To avoid this, we consider rather a k-window abstraction obtained by generating interactions involving exclusively actions belonging to A^i, the actions of the i-th component, where $i \leq k$. The interaction invariant is generated for this k-window. For illustration, we consider the interaction pattern α defined by $\big(a^c, (a_1, o_1), \ldots, (a_m, o_m)\big)$ where the offsets o_i have an ascending order. The k-window $gen(\alpha, k)$ is computed as the disjunction $\cup_{i \in [k]} proj(\alpha^i)$ where $proj(\alpha^i) = \big(a^c, a_1^i, a_2^{i+o_2}, \ldots, a_j^{i+o_j}\big)$. The addition is performed modulo n and j is the last index satisfying that $i + o_j \leq k$. Considering all the interaction patterns, the interaction invariant is computed for $\gamma_k = \cup_{\alpha \in \gamma} gen(\alpha, k)$.

The k-window abstraction is performed in two steps:

1. compute the interaction invariant for the k-window abstraction corresponding to the component C interacting with only k components B^i. The generated invariant is denoted $II_k \stackrel{\triangle}{\equiv} II(\gamma_k)$. We note that the computation of the interaction invariant for this abstraction can be done through the proposed methods of boolean behavioral constraints or linear equations.

   It can be also done through the computation of the global reachable locations of the composed system after abstraction from all timing aspects. In fact, the interaction invariant computation methods aim at over-approximating the reachable locations since its exact computation suffers from the combinatorial explosion problem in case of large systems. However, in the context of small models of systems in our parameterized setting, the computation of this set remains practical and we consider that it can be taken for II_k.

2. the generated invariant is re-indexed by renaming all the indices $j \in [k]$ to $j + i$ in order to get II_k^* of the form $\forall i.II_k[j \leftarrow j + i]$.

Each invariant of the abstraction is also an invariant of the system $C\|_\gamma^n B_i$.

Proposition 7.2. *The formula $(k < n \vee II_k^*)$ is an invariant of $C\|_\gamma^n B_i$.*

Example 7.7. We reconsider the star topology present in the running example in Figure 7.1 and propose to compute a 1-window abstraction. The first step consists in computing the interaction invariant of the system composed of the controller C interacting with an only component B^1. After projection, the interaction set is $II_1 \equiv l_1^c \vee \overline{l_0}[1]$. After step (2), the interaction invariant for $C\|_\gamma^n B_i$ is $II^* \equiv \forall i.l_1^c \vee \overline{l_0}[i]$.

7.4 Parameterizing the Verification Rule

The verification rule becomes, after generalization to the parameterized framework:

$$\underbrace{CI \wedge (k < n \vee II_k^*) \wedge \bar{\mathcal{E}}(\gamma) \wedge \bar{\mathcal{S}}(\gamma)}_{GI} \Rightarrow \Psi \tag{7.5}$$

Showing the validity of the property is equivalent to checking the unsatisfiability of $GI \wedge \neg\Psi$. If Ψ is a T-assertion itself, this formula is also a T-assertion.

Proposition 7.3. *If the property Ψ is a T-assertion, $GI \rightarrow \Psi$ is a T-assertion itself.*

Proof.
Each invariant can be expressed in the prenex form and is a T-assertion.

$$\begin{aligned}
\bar{\mathcal{S}}(\gamma) &\equiv \forall i.\, \wedge_a\, \bar{\mathcal{S}}(a^i)\, \wedge_{a^c}\, \bar{\mathcal{S}}(a^c) \\
&\equiv \forall i.\, \wedge_a \bigwedge_{\substack{(a,o_1)\in\alpha \\ (a,o_2)\in\beta \\ o_1\neq o_2 \vee \alpha\neq\beta}} \left|\overline{h_\alpha}[i-o_1] - \overline{h_\beta}[i-o_2]\right| \geq k_a \wedge \\
&\qquad \forall i_1, i_2.\, \wedge_{a^c} \bigwedge_{\substack{\alpha\ni a^c \\ \beta\ni a^c}} \left|\overline{h_\alpha}[i_1] - \overline{h_\beta}[i_2]\right| \geq k_{a^c} \\
&\equiv \forall i, i_1, i_2.\, \wedge_a \bigwedge_{\substack{(a,o_1)\in\alpha \\ (a,o_2)\in\beta \\ o_1\neq o_2 \vee \alpha\neq\beta}} \left|\overline{h_\alpha}[i-o_1] - \overline{h_\beta}[i-o_2]\right| \geq k_a \\
&\qquad \wedge_{a^c} \bigwedge_{\substack{\alpha\ni a^c \\ \beta\ni a^c}} \left|\overline{h_\alpha}[i_1] - \overline{h_\beta}[i_2]\right| \geq k_{a^c}
\end{aligned}$$

Given the set of the controller's actions $A^c = \{a_1, \ldots, a_m\}$, $\exists j_{A^c}$ denotes the prefix $\exists j_{a_1} j_{a_2} \ldots j_{a_m}$. The index j_{a^c} stands for an arbitrary element in j_{A^c}.

$$\begin{aligned}
\bar{\mathcal{S}}(\gamma) &\equiv \forall i\, \wedge_a\, \mathcal{E}(a^i)\, \wedge_{a^c}\, \mathcal{E}(a^c) \\
&\equiv \forall i_1, i_2. \exists j_{A^c}. \left(\bigvee_{(a,o)\in\alpha} \overline{h_a}[i_1] = \overline{h_\alpha}[i_1 - o]\right) \wedge \left(\bigwedge_{(a,o)\in\alpha} \overline{h_a}[i_1] \leq \overline{h_\alpha}[i_1 - o]\right) \wedge \\
&\qquad \wedge_{a^c} \left(\bigvee_{a^c\in\alpha} h_{a^c} = \overline{h_\alpha}[j_{a^c}]\right) \wedge \left(\bigwedge_{a^c\in\alpha} h_{a^c} \leq \overline{h_\alpha}[i_2]\right) \qquad (7.6)
\end{aligned}$$

Given that the quantified variables are not shared between the different invariants, the following basic equivalences can be used to convert $GI \Rightarrow \Psi$ into a T-assertion form.

$$\begin{aligned}
&\mathbf{Q}x\mathbf{Q}y.(P(x)\ op\ R(y)) \equiv \mathbf{Q}y\mathbf{Q}x.(P(x)\ op\ R(y)) \\
&P\ op\ \mathbf{Q}y.R(y) \equiv \mathbf{Q}y.(P\ op\ R(y))
\end{aligned}$$

op denotes any logical connective and **Q** any quantifier.
Since $GI \Rightarrow \Psi$ can be written under T-assertion form, the small model theorem expressed in Proposition 7.1 can be applied.

Corollary 7.1. *Given a PTS $C\|_\gamma^n B_i$ and a property $\Psi \stackrel{\triangle}{=} \forall\bar{s}\exists\bar{t}.\Psi^\circ$, it suffices to check the validity of $\neg GI \vee \Psi$ for $n \leq \#\bar{s} + \#A^c + 2$ in order to assert the validity of Ψ for any n.*

Example 7.8. For illustration, we summarize the obtained T-assertion for the running example. The considered safety property is $\Psi \stackrel{\triangle}{=} \exists i.x^c = \overline{x}[i]$. It expresses

that one of the clocks in $\bar{x}$ has the same value as x^c. We have already gone through the conversion of the invariants into T-assertions, expressed in Examples 7.5, 7.6 and 7.4. What is left is to combine them and rename the quantified variables to obtain:

$$\forall i\ \exists j_1, j_2, j_3, j_4, j_5. \Bigg(\neg \Big((l_0^c \wedge h_{a^c} \geq 0 \wedge x^c \geq 0 \vee l_1^c \wedge x^c = h_{a^c} \geq 0) \wedge \\ (\overline{l_0}[j_1] \wedge \overline{h_a}[j_1] \geq 0 \wedge \overline{x}[j_1] \geq 0 \vee \overline{l_1}[j_1] \wedge \overline{x}[j_1] = \overline{h_a}[j_1] \geq 0) \wedge \\ (\overline{h_a}[j_2] = \overline{h_\alpha}[j_2] \geq h_{a^c}) \wedge (h_{a^c} = \overline{h_\alpha}[i]) \wedge \left| \overline{h_\alpha}[j_3] - \overline{h_\alpha}[j_5] \right| \geq k_{a^c} \Big) \vee \\ x^c = \overline{x}[j_4] \Bigg) \tag{7.7}$$

Above, we have used the component invariants with respect to the extensions with history clocks. By applying Corollary 7.1, we can assert the correctness of Ψ from the validity the formula (7.7) for $n \leq 3$ processes.
By application of Corollary 7.1, the validity of Ψ can be asserted from the validity of the formula (7.7) for every number $n \leq 3$.

7.5 Towards Efficient Verification of Parameterized Timed Systems

The present verification approach for parameterized timed systems states that if the global invariant implies the property for the small models, then it is verified for all numbers of replicas. We want to complete it in order to cover the cases where the refinement of the invariant with the negation of the spurious counterexamples is needed. Since the backward reachability computation is in general practical for small models, this extension of our uniform verification method would drastically reduce the verification time even when raised false positives are eliminated. The main issue concerns the generalization of the concrete real valuations relative to a given counterexample to a generic characterization which ensures convergence. We suppose that all the counterexamples generated for small models are proven to be spurious, meaning that the property is valid. We denote by $\mathcal{D}$ the set of the analyzed counterexamples under the generalized form. The final refined global invariant is expressed as follows:

$$GI_r \equiv GI \wedge \bigwedge_{(l,\zeta)\in\mathcal{D}} \neg(at(l) \wedge \zeta)$$

where GI is the initial invariant, that is the global invariant before refinement with the counterexamples negation. If we prove that $GI_r \wedge \neg\Psi$ can be always expressed in the T-assertion grammar, then the use of the small model theorem could be allowed even in case the spurious counterexamples are eliminated. This requires a deep analysis of the generalized counterexamples that depend from the literals appearing in the safety property and which they negate. Each counterexample (l, ζ) in $\mathcal{D}$ is deduced from a valuation of the variables generated by the Sat-solver as a solution to $GI \wedge \neg\Psi$.
The refined global invariant negates the set of spurious counterexamples.

$$\begin{aligned}\neg(at(l) \wedge \zeta) &\equiv \neg(at(l) \wedge \bigwedge_{z_k \in \mathcal{L}.\nu \models z_k} z_k \wedge \bigwedge_{z_k \in \mathcal{L}.\nu \not\models z_k} \neg z_k) \\ &\equiv \neg at(l) \vee \bigvee_{z_k \in \mathcal{L}.\nu \models z_k} \neg z_k \vee \bigvee_{z_k \in \mathcal{L}.\nu \not\models z_k} z_k\end{aligned}$$

The set $\mathcal{L}$ contains the literals forming the Ψ property. The property itself can be expressed as a T-assertion for the parameterized setting. To show that the new refined invariant can be also expressed as a T-assertion, symmetry techniques could be useful to express the counterexamples set $\mathcal{D}$ as a generic characterization. We give a brief clue for having such generic characterization. We suppose that all the analyzed counterexamples for the small models are proven to be spurious by use of the backward reachability computation. For a fixed number of processes, let it be the number of universal quantifiers in $GI \wedge \neg\Psi$, the generation of the counterexamples and the refinement of the global invariant GI is iteratively performed until no counterexample remains.

1. For a generated counterexample θ, the generalization with respect to the literals of the safety property is first performed.
2. In in the next iteration, before The Sat-solver proceeds to the satisfiability checking of $GI_r \wedge \neg\Psi$, the global invariant GI_r is refined not only with the counterexample example (l, ζ), but also with the set of symmetric states.
3. Depending from the safety property and the generated states that negate it, the final set GI_r could be generalized as a T-assertion.

The formalization of the above steps requires further investigation. In the following, we give an intuition by showing two manually processed examples.

Example 7.9. 1. We consider a simple case for illustration purpose. Let the safety property be $\Psi \stackrel{\triangle}{\equiv} at(lc_1) \wedge \bigwedge_i at(l_1^i) \Rightarrow \exists j.x_j - y \geq 0$. The generated counterexample is generalized with respect to the literals in Ψ.
The negation of the safety property is $\neg\Psi \equiv at(lc_1) \wedge \bigwedge_i at(l_1^i) \wedge \forall j.x_j - y < 0$.

Considering the universal quantifier $\forall j$ on the zones and the fact that $\neg\Psi$ precises the locations of all the components, there is an only configuration for generalized counterexample that may satisfy it. In the parameterized setting, the refined global invariant can be easily expressed as a T-assertion

$$GI_r \equiv GI \wedge \neg\big(at(lc_1) \wedge \forall i.(at(l_1^i) \wedge x_i - y < 0)\big)$$

2. We consider a more complicated case where the safety property is

$$\Psi \stackrel{\triangle}{\equiv} at(lc_1) \wedge \bigwedge_i at(l_1^i) \Rightarrow \forall i_1, i_2.(x_{i_1} - x_{i_2} \geq 4 \vee x_{i_2} - x_{i_1} \geq 4)$$

We suppose that the first counterexample θ generated by the Sat-solver for $k = 4$ corresponds to the following generalization:

$$\begin{aligned} generalize(\theta, \Psi) = ((lc_1, l_1^1, l_1^2, l_1^3, l_1^4), & x_1 - x_2 < 4 \wedge x_2 - x_1 < 4 \\ & \wedge x_1 - x_3 \geq 4 \wedge x_3 - x_1 < 4 \\ & \wedge x_3 - x_4 \geq 4 \wedge x_4 - x_3 < 4 \\ & \wedge x_1 - x_4 \geq 4 \wedge x_4 - x_1 < 4) \end{aligned}$$

Using symmetry arguments, we can deduce that the following set contains symbolic states that do also violate the safety property:

$$\begin{aligned} Symm(generalize(\theta, \Psi)) = \{(lc_1, l_1^1, l_1^2, l_1^3, l_1^4), & \exists i_1, i_2 \,.\, x_{i_1} - x_{i_2} < 4 \wedge x_{i_2} - x_{i_1} < 4\} \\ & \wedge \forall j \neq i_1, i_2 \,.\, x_{i_1} - x_j \geq 4) \end{aligned}$$

This set can be iteratively built by iterating on the analyzed counterexamples, but the use of symmetry may help finding a formulation of the final refined invariant as a T-assertion.
We denote by GI_4 (resp. $GI_{r,4}$) the global (resp.refined global) invariant for this precise number 4 of replicas, whereas the global invariant before and after refinement are denoted in the parameterized setting by GI and GI_r respectively.

$$GI_{r,4} \equiv GI_4 \wedge \bigwedge_{(l,\zeta) \in Symm(generalize(\theta,\Psi))} \neg(at(l) \wedge \zeta)$$

In order to check if $GI_{r,4}$ refutes all the possible counterexamples, we proceed to the satisfiability checking of $GI_{r,4} \wedge \neg\Psi$. If no solution exists, the new invariant $GI_{r,4}$ is strong enough to detect the property. If other solutions exist, we try to express them similarly as *families* of symmetric states. By finding an

expression GI_r generalized from $GI_{r,4}$ to a T-assertion and implying the safety property for all numbers of replicas in the small models, the application of the small model theorem becomes possible. In some way, the negation of the spurious counterexamples is a new type of invariant that we want to express also as a T-assertion in the parameterized setting. For the current example:

$$GI_r \equiv GI \wedge \neg\big(at(lc_1) \wedge \forall i.(at(l_1^i) \wedge \exists i_1, i_2.x_{i_1} - x_{i_2} < 4 \wedge x_{i_2} - x_{i_1} < 4\} \\ \wedge \forall j \neq i_1, i_2.x_{i_1} - x_j \geq 4)\big)$$

In order to apply the small model theorem to $GI_r \wedge \neg\Psi$, the negated counterexamples should be proven to be invalid for the small models. The cutoff for which the invalidity of the counterexamples should be proven through backward analysis is deduced from the number of universal quantifiers in $GI_r \wedge \neg\Psi$.

7.6 Summary

The proposed compositional verification method supports an elegant application of the small model theorem aiming to address the problem of uniform verification of parameterized timed systems. In fact, the global invariant gathers local invariants of its components implying similar characterizations among the replicas. Besides, the modeling framework benefits from correspondences between typical interaction structures and topologies.

Given that all the compositionally computed invariants can be expressed as T-assertions, the application of this small model theorem is direct. However, in case where spurious counterexamples exist, it is required that the refined global invariant, obtained by elimination of these counterexamples, can be expressed as a T-assertion. A further investigation in this direction is required in order to demonstrate that the negation of the spurious counterexamples can be always generalized to a T-assertion, which highly depends on the shape and the literals of the safety property of interest.

The application of the small model theorem is restricted to systems of timed automata and is not applicable to timed automata parametric in their indices. An extension in this direction would be of interest.

Chapter 8

Implementation: RTD-Finder

Contents

We present in this chapter RTD-Finder, a tool for the verification of safety properties for real-time systems which implements our compositional invariant generation method. The toolset contains the extension aimed to analyze the possible counterexamples. First, we present the structure of RTD-Finder and briefly introduce the RT-BIP language. Afterwards, we detail the implementation of the different RTD-Finder modules.

8.1 The RTD-Finder Tool

The tool takes as input a file modeling a timed system in the RT-BIP language. It is implemented in the Java programming language and interacts with the Yices Sat-Solver.

8.1.1 The RT-BIP Language

BIP (Behavior, Interaction, Priority) is a component-based language for modeling and programming complex systems. It offers a practical implementation of the three layers of BIP , Behavior, Interactions and Priorities, introduced at Chapter 3. The RT-BIP language [ACS13] is the timed extension of the BIP language supporting the main timing features introduced in Chapter 4. RT-BIP was initially designed to support the timing constraints on transitions as time intervals with urgency types. In this type of models, the time progress at a given location depends from the urgency level of the transitions enabled from that location. In particular, time is not allowed to progress at a location if there exists an outgoing *urgent* transition that is enabled.
In the framework of thesis, we don't work on the RT-BIP model with urgencies. Instead, we work on another version of RT-BIP supporting time progress conditions on locations as defined in Chapter 4.
Compared to the untimed BIP framework, the main difference of RT-BIP lies on the behavior layer which is extended to timed automata (and timed Petri nets). In Appendix A, we recall the main constructs of the BIP language and illustrate the syntax of RT-BIP by modeling the timed Controller-Workers system depicted in Figure 4.5.

8.1.2 RTD-Finder Structure

The structure of RTD-Finder is depicted in Figure 8.1. The tool takes as input a RT-BIP source file and a safety property Ψ to check for invariance. If the property is not provided by the user, the tool proceeds by default to the verification of deadlock-freedom. Following this, it computes the predicate characterizing the set of deadlock states in the so-called DLK module. The tool proceeds basically as follows:

1. The tool extends each component B_i from the input file with history clocks (HC) into B_i^h.

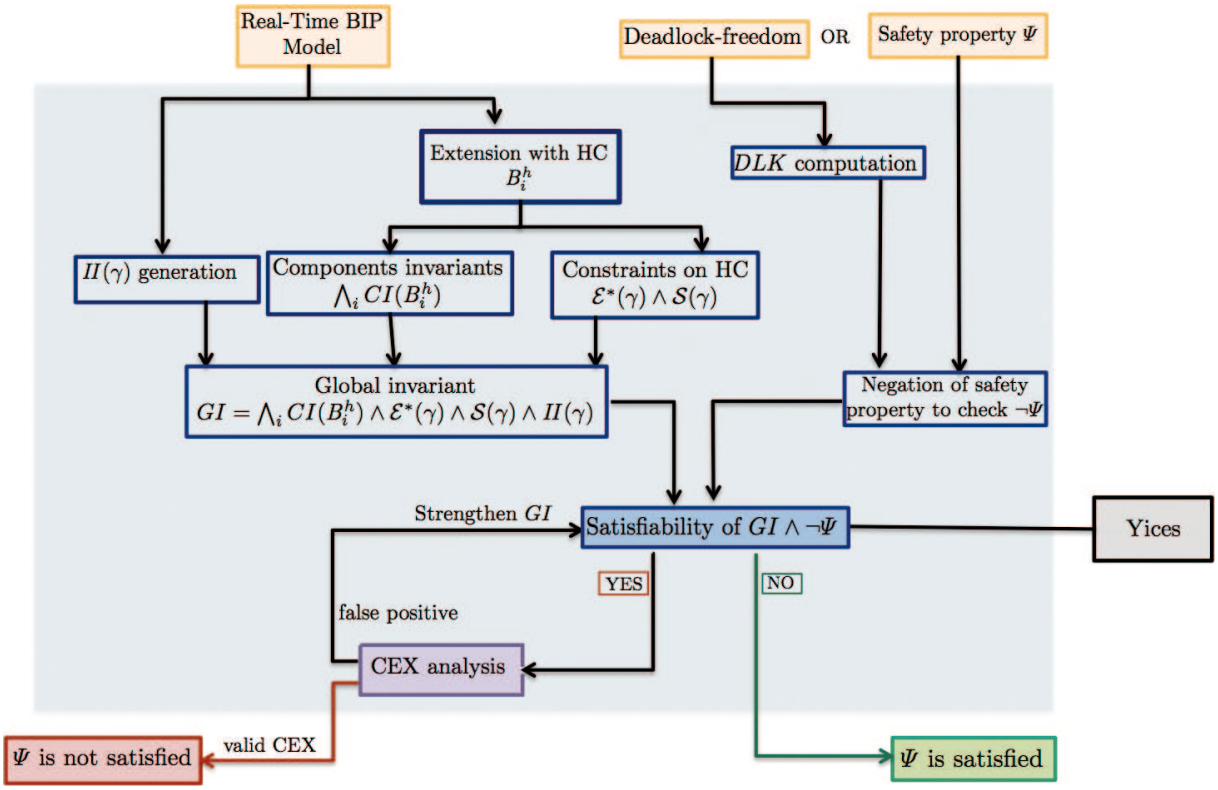


Figure 8.1: The RTD-Finder tool structure

2. It then computes the invariants of B_i^h as the set of reachable symbolic states. These, in turn, are computed by means of a DBM library that we developed.

3. It computes the interaction invariant

4. The inequalities on history clocks ($\mathcal{E}^*(\gamma)$ and $\mathcal{S}(\gamma)$) are computed.

5. The combination of all the above invariants forms the global invariant GI^h.

6. Together with the property Ψ, this invariant is input to Yices [DdM06] [Dut14], an SMT solver.

7. If $GI^h \wedge \neg\Psi$ is unsatisfiable, the property is valid. Else, a counter-example is generated. A backward analysis module is developed to decide upon their validity (the dashed box in Figure 8.1).

In the following, more details are given for each of the above steps. We note that the implementation of the interaction invariant computation method and the data variables consideration through abstraction are not shown since a detailed presentation can be found in [Ngu10].

8.2 Invariants Construction

8.2.1 Component Invariant Generation

8.2.1.1 Difference Bound Matrices Library Implementation

As explained in Chapter 4, the operations on zones are defined in terms of clocks assignments and DBM offer a convenient data structure to implement them. In [BY03], most of the common operations on zones were detailed together with pseudo code for their implementation. In the framework of this thesis, a DBM library was implemented in the Java programming language to provide the requisite operations on zones that are used in RTD-Finder. We mention clock reset and its inverse operation, the extrapolation, the normalization, the addition, the intersection, Floyd-Warshall algorithm, the inclusion test and the forward and downward operations. In this section, we give pseudo codes for the implementation of some operations on DBM which were not presented in [BY03]. The function "exists" shown in Algorithm 1 computes the zone resulting from the deletion of a precise clock by omitting all the clocks constraints involving it and keeping the remaining clocks and constraints relating them. In RTD-Finder, this function serves to perform the inclusion test over zones independently from the amount of time h_0 elapsed since the beginning, whereas the function "msol" shown in Algorithm 2 is used to compute the minimum time elapse required between two occurrences of the same action as explained in Chapter.5.

8.2.1.2 Reachability Graph Computation

The local invariant implementation for an atomic component $B = (L, A, \mathcal{X}, T, \mathsf{tpc}, s_0)$ lays upon the computation of the reachability graph. This step is preceded by the extension of the component with the history clocks. An action is *exported* if it is synchronized with actions of other components. In the opposite case, it is *internal*. By A_{exp} we refer to the actions in A which are exported.
The reachability graph computation is shown in Algorithm 3 and starts from the initial state of the component, defined by its initial location, its time progress condition and the initialization of the clocks, including the history clocks. Afterwards, the Depth-First-Search (DFS) graph strategy is performed to find the reachable symbolic states. The symbolic states who are still to be explored are organized in a stack *toVisit* while all the already visited symbolic states are added to the set *visited*. The set *visited* is filled as long as the stack *toVisit* is not empty. The

Algorithm 1: function exists(z, x) ;

Input : A zone z having M for DBM and a clock x

Output: z' computed from z by totally omitting constraints on the clock x

```
1  Let C be the ordered set of clocks of M ;
2  n ← length(C) ;
3  if z does not contain x then
4  |  return z ;
5  end
6  else
7  |  k ← 0 ; r ← 0 ; c ← 0 ;
8  |  for i ← 0 to n do
9  |  |  if C(i) ≠ x then
10 |  |  |  C'(k) ← C(i) ;
11 |  |  |  k++ ;
12 |  |  |  for j ← 0 to n do
13 |  |  |  |  if C(j) ≠ x then
   |  |  |  |  |  // Filling in the elements of the new DBM
14 |  |  |  |  |  m'_rc ← m_ij ;
15 |  |  |  |  |  c++ ;
16 |  |  |  |  end
17 |  |  |  end
18 |  |  |  r++ ;
19 |  |  |  c ← 0 ;
20 |  |  end
21 |  end
22 end
23 return the zone z' having for DBM M' with elements m'_ij and clocks C' ;
```

Algorithm 2: function msol((z, x)) ;

Input : A zone z having M for DBM, and a clock x.
Output: The minimal solution over x to the constraints in z.

```
1 Let i be the index of clock x in M ;
2 if m_0i ≠ ∞ then
  |  //   ∃c. m_0i = (≺, c)
3 |  return c ;
4 end
5 else
6 |  return 0 ;
7 end
```

latter is filled in with the successors of each new visited state. To this purpose, a test is performed over each new candidate symbolic state (Line 9 to 12). If a symbolic state s has been already added to the *visited* set or if there exists a state s_v in *visited* which has the same location and whose zone includes the zone of s, then s is considered visited as well. Its successors are not computed nor added to the stack. The inclusion test on the zones (Line 10) is performed independently from the value of h_0. The function *exists* on zones is implemented in the DBM library at this precise intent.

The function "getSuccessors" computes the set of the successors of a symbolic state $s = (l, \zeta)$ and is shown in Algorithm 4. To this purpose, it computes the successor with respect to each transition t having l for source location. The zone of the successor state is computed by consideration of the guard of the transition, the reset of the clocks and the time progress condition of the destination location. Its computation is detailed in Algorithm 5. The obtained zone is extrapolated with the maximum constant appearing in the component and is afterwards normalized (Line 5).
The minimum time delay k_a between two consecutive occurrences of the same action a is dynamically computed during this step. In fact, when a new symbolic state is reached through transition t labeled with a action, the minimum time elapse k_a is updated if required. This represents an application of the dynamic programming solution introduced in Section 5.2 and implemented in Algorithm 6. We recall that the value $msol(\zeta, h_0)$ is the minimal solution over h_0 to the constraints in the zone ζ. This function is implemented in the DBM library and is presented in Algorithm 2.

Algorithm 3: function computeReachGraph() ;

Input : An atomic component B^h extended from $B = (L, A, \mathcal{X}, T, \mathsf{tpc}, s_0)$ where $s_0 = (l_0, tpc_0)$.

Output: The set of reachable states of B^h.

```
1  Set of states visited ← ∅ ;
2  Stack of states toVisit ← ε ;
3  boolean isVisited ← false ;
4  ζ0 ←↗ (tpc0 ∧ ⋀x∈X x = 0 ∧ h0 = 0 ∧ ⋀a∈Aexp ha = 0);
5  toVisit.push((l0, ζ0)) ;
6  while toVisit ≠ ε do
7      s ← toVisit.pop() ;
8      isVisited ← false ;
9      for each state sv = (lv, zv) in visited do
10         if lv = l and exists(sv, h0) ⊆ exists(s, h0) then
11             isVisited ←true ;
12         end
13     end
14     if isVisited is false then
15         visited ← visited ∪ {s} ;
16         for each state suc in getSuccessors(s) do
17             toVisit.push(suc) ;
18         end
19     end
20 end
21 return visited;
```

Algorithm 4: function getSuccessors(s) ;

Input : A symbolic state $s = (l, \zeta)$ of the atomic component B^h

Output: The set of successors of s state

for *each transition* $t = (l, (a, g, r), l')$, *having* l *for source location* **do**
  $\zeta' \leftarrow \zeta$;
  if $l \neq l'$ *or* $g \neq \emptyset$ *or* $r \neq \emptyset$ **then**
    $(l', \zeta') \leftarrow succ(s, t)$;
    $successors \leftarrow successors \cup \{(l', \zeta')\}$;
  end
  updateMinTimeDelay$(msol(\zeta', h_0), a)$;
end
return *successors*;

Algorithm 5: function succ(s, t) ;

Input : A transition $t = (l, (a, g, r), l')$, a symbolic state (l, ζ).

Output: The successor symbolic state of s through t.

$\zeta' \leftarrow \zeta \wedge g$;
for *each clock* x *to reset in* r **do**
  $\zeta' \leftarrow \zeta' [x \leftarrow 0]$;
end
if a *is exported* **then**
  $\zeta' \leftarrow \zeta' [h_a \leftarrow 0]$;
end
$\zeta' \leftarrow \nearrow (\zeta' \wedge tpc(l')) \wedge tpc(l')$;
return $(l', \mathsf{extrap}_{k_{max}}(\zeta'))$;

Algorithm 6: function updateMinTimeDelay(occ, a) ;

Input : occ: the time elapsed since the beginning to reach a new occurrence of a in component B.

Output: The constant k_a relative to action a of the component is updated.

$i++$;
$oldOcc \leftarrow newOcc$;
$newOcc \leftarrow occ$;
if $0 \leq newOcc - oldOcc < k_a$ *and* $i > 1$ **then**
  $k_a \leftarrow newOcc - oldOcc$;
end

Algorithm 7: function ComputeHCconstraints() ;

Input : a system with interaction set $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$.

Output: the constraints relating the history clocks.

Initialization: $\mathcal{E} \leftarrow$ **true**, $\mathcal{S} \leftarrow$ **true** ;
for *each exported action* a **do**
  $\mathcal{E}_{a,\leq} \leftarrow$ **true** ; $\mathcal{E}_{a,eq} \leftarrow$ **false** ; $\mathcal{S}_a \leftarrow$ **true** ;
  for *each interaction* α *containing* a **do**
    $\mathcal{E}_{a,\leq} \leftarrow \mathcal{E}_{a,\leq} \wedge h_a \leq h_\alpha$;
    $\mathcal{E}_{a,eq} \leftarrow \mathcal{E}_{a,eq} \vee h_a = h_\alpha$;
  end
  $\mathcal{E}_a \leftarrow \mathcal{E}_{a,eq} \wedge \mathcal{E}_{a,\leq}$;
  $\mathcal{E} \leftarrow \mathcal{E} \wedge \mathcal{E}_a$;
  if a *is conflicting and* $k_a > 0$ **then**
    for *each* α_i *containing* a **do**
      for *each* α_j *containing* a **do**
        if $i < j$ **then**
          $\mathcal{S}_a \leftarrow \mathcal{S}_a \wedge (h_{\alpha_j} \geq h_{\alpha_i} + k_a \vee h_{\alpha_i} \geq h_{\alpha_j} + k_a)$;
        end
      end
    end
  end
  $\mathcal{S} \leftarrow \mathcal{S} \wedge \mathcal{S}_a$;
end
return $\mathcal{E} \wedge \mathcal{S}$;

The update of k_a constant stops when the computation of the reachability graph is completed, that is when no new symbolic state can be reached. We note that i is initialized to 0 once for each a action, before the function call, while k_a is initialized to k_{max}.

8.2.2 History Clocks Constraints Generation

The history clocks constraints $\mathcal{E}(\gamma)$ and $\mathcal{S}(\gamma)$ allow to express the relations between the history clocks which serve as auxiliary clocks. New relations between the original clocks of the different components are induced from the history clocks constraints. Their computation is illustrated in Algorithm 7. The resulting $\mathcal{E}$ and $\mathcal{S}$ predicates are directly written under Yices format to the output file.

8.3 DIS Generation

Algorithm 8: function generateDIS($\gamma, \|_\gamma B_i$) ;

Input : A timed system $\|_\gamma B_i$.
Output: The predicate characterizing the deadlocked states.

```
1 Initialization: en ← false ;
2 for each interaction α ∈ γ do
3 |   en ← en ∨ getEnabledStates(α) ;
4 end
5 DIS ← ¬en ;
6 return DIS ;
```

The implementation of the deadlock predicate generation module requires the computation of the enabled global transitions as explained in Section 4.4.2. The system is deadlocked if none of the interactions is enabled as presented in Algorithm 8. The enabledness of a global transition labeled α requires that all the participating actions can be executed at the same time. For this aim, the function $getLocalZoneMap(a)$ is used to build a map giving for each location l of the component the set of zones relative to the transitions labeled with a and having for source location l. Therefore, the total number of zones mapped to l is equal to the number of outgoing transitions. We take into account that many local transitions may be labeled with the same action a and have the same source location l. This is allowed since the components can be non-deterministic. The

value mapped to such a location should contain the zones relative to all of these transitions. As shown in Algorithm 9, for each transition t, the computation of the zone requires the consideration of the time progress condition of the destination location, the reset operations and the guard of the transition (Line 3). However, the time progress condition of the source location is considered later, in the function $getEnabledStates(\alpha)$ since the backward operation should be performed globally.

The generation of the predicate $enabled_\alpha$ characterizing the enabledness of a given interaction α is presented in Algorithm 9. At the system level, a map $\mathcal{M}$ is used to store all the zones relative to all the locations in the different components. A given location may be the source of many transitions triggered by various interactions, requiring that the values in $\mathcal{M}$ are cumulative (Line 5 of Algorithm 9).

In the case of a component which is not involved in the α interaction, meaning that it remains at the same location, we map the zone set $\{tpc(l)\}$ for every location l that it contains. Besides, their time progress conditions should be considered while computing the zone of the global enabled states expressed in Line 21. We note that in order to build the predicate $enabled_\alpha$ expressing the enabled global states set, the cartesian product operation is required to get the global locations, picking a location from each component, as well as to extract the related zones.

8.4 Satisfiability Checking

The satisfiability checking is required to verify if the global invariant implies the safety property. The unsatisfiability of $GI^h \wedge \neg\Psi$ implies the validity of the property for all the states of the system. Since the predicate GI^h is defined over the clocks, represented as real-valued variables, the satisfiability checking by use of the CUDD package for manipulating the Binary Decision Diagrams (BDDs) is not possible. Instead, the unsatisfiability checking is performed by use of the Sat-solver Yices. Yices 2 is an SMT (SAT modulo theories) solver that decides the satisfiability of formulas containing uninterpreted function symbols with equality, linear integer and real arithmetic, bitvectors, scalar types, and tuples. Given a formula f, Yices checks whether there exists a configuration of the defined variables that satisfies f. If such a solution exists, Yices generates a `sat` output together with the values of the variables in the generated solution. If not, an `unsat` output is produced.
For illustration, we show how the invariants computed in Section 4.6 are encoded in Yices. For each component, the locations are first defined as boolean variables

Algorithm 9: function getEnabledStates(α) ;

Input : An interaction α.

Output: The predicate characterizing the set of global states enabling α

Initialization: $\mathcal{M}$ an empty map, $\mathcal{L} = \{\{\}\}$ an empty set of sets of locations ;
$enabled_\alpha \leftarrow$ **false** ;
for *each action* $a \in \alpha$ **do**
  $\mathcal{M}_a \leftarrow getLocalZoneMap(a)$;
  $putAllCumulative(\mathcal{M}, \mathcal{M}_a)$;
  $\mathcal{L} \leftarrow \mathcal{L} \cup getKeySet(\mathcal{M}_a)$;
end
for *each component* $B = (L, A, \mathcal{X}, T, \mathsf{tpc}, s_0)$ *non involved in* α **do**
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{L\}$;
  for *each location* $l \in L$ **do**
    $map(\mathcal{M}, (l, \{tpc(l)\}))$;
  end
end
$\mathcal{L}_{global} \leftarrow \mathsf{X}_i \ \mathcal{L}(i)$
for *each global location* $\bar{l} \in \mathcal{L}_{global}$ **do**
  $\mathcal{Z}_{global} \leftarrow \mathsf{X}_{l \in \bar{l}} \ get(\mathcal{M}, l)$;
  $Z_{en} \leftarrow \{\}$;
  for *each zone set* $\bar{Z}$ *in* $\mathcal{Z}_{global}$ **do**
    $Z_{en} \leftarrow Z_{en} \cup \{\bigwedge_{z \in \bar{Z}} z\}$;
  end
  $enabled_\alpha \leftarrow enabled_\alpha \vee (at(\bar{l}) \wedge tpc(\bar{l}) \wedge \bigvee_{z \in Z_{en}} \swarrow z)$
end
return $enabled_\alpha$

Algorithm 10: function getLocalZoneMap(a) ;

Input : An action a.

Output: The map $\mathcal{M}$ matching to each location the set of successor zones

for *each transition* $t = (l, (a, g, r), l')$, *labeled with* a **do**
  $z \leftarrow g \wedge tpc(l')[r]$;
  if l *already belongs to the keys of* $\mathcal{M}$ **then**
    $get(\mathcal{M}, l) \leftarrow get(\mathcal{M}, l) \cup \{z\}$
  end
  else
    $map(\mathcal{M}, (l, \{z\}))$
  end
end
return $\mathcal{M}$

Algorithm 11: function putAllCumulative$(\mathcal{M}, \mathcal{M}_a)$;

Input : Two maps $\mathcal{M}$ and $\mathcal{M}_a$.

Output: Update the values of $\mathcal{M}$ with the values of M_a.

for *each key location* l *in* $\mathcal{M}_a$ **do**
  if l_a *already belongs to the keys of* $\mathcal{M}$ **then**
    $get(\mathcal{M}, l) \leftarrow get(\mathcal{M}, l) \cup get(\mathcal{M}_a, l)$;
  end
  else
    $map(\mathcal{M}, (l, get(\mathcal{M}_a, l)))$;
  end
end

and the clocks are defined as real variables. These definitions are followed by the formulation of the component invariants. For example, the component invariant of `worker1` is denoted by `worker1CI`. After the components invariants, the interaction invariant `II` and the history clocks constraints are presented, followed by the global invariant which is defined as their conjunction. The negation of the safety property is formulated in the `notSafe` predicate.
The three last lines give an order to Yices to check the satisfiability of $GI \wedge \neg\Psi$ and to generate a solution if exists.

```
(define y1 :: real)
(define l1worker1::bool)
(define l2worker1::bool)

(define worker1CI :: bool (and (or   l1worker1
                                     ( and l2worker1   (  >=   y1  4 )  )
                               )
                          ( >=   y1  0 )
                          (not  (and l1worker1 l2worker1 ) )
                          ) )

(define x :: real)
(define lc0controller::bool)
(define lc1controller::bool)
(define lc2controller::bool)

(define controllerCI :: bool (and (or lc0controller
                                      (and lc1controller (<= x 4))
                                      lc2controller )
                                  (>= x 0)
                                  (not (or (and lc0controller lc1controller )
                                           (and lc0controller lc2controller )
                                           (and lc1controller lc2controller )
                                    ) )
                                 ) )

(define II :: bool  ( and (or l1worker1 lc2controller )
                          (or l2worker1 lc0controller lc1controller )
                     ))

(define GI :: bool (and worker1CI  controllerCI II )  )

(define notSafe :: bool (and (and lc1controller  l1worker1 )
                             (< y1  x ) ) )

(assert (and GI   notSafe ))
(check)
(show-model)
```

The above example does not apply our compositional verification method which lays upon introducing the history clocks and asserting the constraints relating them as invariants and shows a direct application of the invariant computation method in the D-Finder tool. It results that the above global invariant `GI` fails to capture the safety property of interest and the output generated by Yices is:

```
sat
(= l2worker1 false)
```

```
(= lc0controller false)
(= lc2controller false)
(= l1worker1 true)
(= lc1controller true)
(= y1 3)
(= x 4)
```

The generated solution satisfies the global invariant and nonetheless contradicts Ψ since $y_1 - x = -1 < 0$.
RTD-Finder computes the global invariant by use of the verification method presented in Chapter 5. To this purpose, it defines the history clocks in the Yices file as real variables and the related invariants as boolean variables conjoined to the component and interaction invariants in order to make stronger GI.

8.5 Counterexample Analysis for Global Invariant Refinement

The counterexample analysis module implements the algorithm presented in Figure 6.1. For each execution, The Sat-solver Yices generates a solution to the predicate $GI^h \wedge \neg\Psi$. The generated counterexample is analyzed in order to check its validity. If it is spurious, the global invariant GI^h is strengthened with its negation. The generated counterexamples are iteratively parsed until Yices testifies the unsatisfiability of $GI^h \wedge \neg\Psi$, or until a generated counterexample is proven to be reachable. The first step for the analysis of a given counterexample consists in parsing the solution generated by Yices and satisfying $GI^h \wedge \neg\Psi$. Secondly, the generalization of the zone of the counterexample is performed based on the atomic constraints appearing in the safety property Ψ. The DBM library is used at this aim. Thirdly, the backward reachability computation is performed in order to check the validity of the generalized counterexample.

8.6 Summary

In this chapter, we gave an overview of the implementation of the RTD-Finder tool for checking safety properties of real-time systems modeled in the RT-BIP language. We first described the structure of the tool, followed by the implementation details of the different modules.
In the next chapter, we present the experimental results for different benchmarks,

reflecting the efficiency of the compositional invariant generation method as well as its scalability.

Chapter 9

Experimentation

Contents

This chapter shows several case-studies verified using the RTD-Finder tool. The classical Train-Gate-Controller and Token Ring systems are first described and verified. Afterwards, we introduce three other benchmarks which are represented by a temperature control system, a gear controller system and a dual chamber implantable pacemaker. A comparison of the verification time is carried out between RTD-Finder and the UPPAAL tool [BDL+06]. The experimental results witness the efficiency of the proposed invariant generation method in many cases as well as the scalability of the method, especially when no counterexample is generated.

9.1 Train-Gate-Controller System

We note that all the experiments are run on Linux machine Intel Core 3.20 GHz $\times$4 and 15.6 GiB Ram.

The first example is the classical *Train-Gate-Controller* (TGC) system, where a controller, a gate and a number of trains interact together. For simplicity, in Figure 9.1 we show only the gate and the controller interacting with one train. The controller ensures that the gate is down when one of the trains enters the crossing. It also ensures that when all the trains are far away, the gate is raised. We note that when there is more then one train, the actions *approach* and *exit* of the controller are conflicting. They are shared between the interactions $approach_i \mid approach$ (respectively $exit_i \mid exit$).

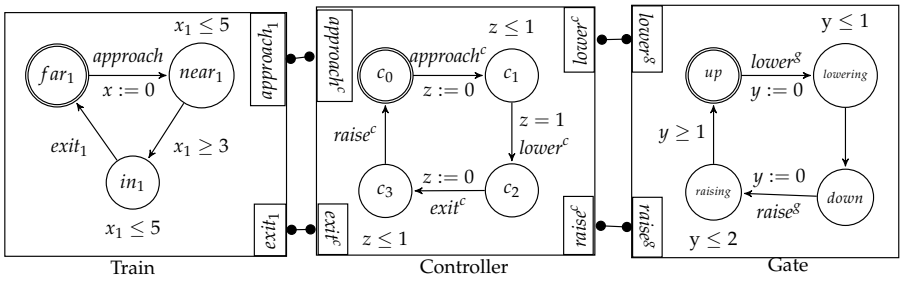


Figure 9.1: A controller interacting with a train and a gate

We verified two properties:

1. (P_1) Utility property: The gate does not go down if all the trains are far from the crossing.

$$P_1 \equiv \bigwedge_i at(Far_i) \Rightarrow \neg at(low)$$

Trains number	Property	n	q	c	$\lvert\gamma\rvert$	h	t	t_{yices}
50	P_1	52	158	52	102	106	$0.5s$	$0.3s$
100	P_1	102	308	102	202	206	$5.3s$	$0.6s$
200	P_1	202	608	202	402	406	$1m33s$	$5s$
300	P_1	302	908	302	602	606	$9m8s$	$20s$
500	P_1	502	1508	502	1002	1006	$1h13m20s$	$2m52s$

Table 9.1: Verification results for P_1 property

The generated invariant was strong enough to verify this utility property. The verification results are depicted in Table 9.1. In this table, n is the number of components in the considered example, q is the total number of control locations of its components and c (resp. h) is the number of system clocks (resp. actions history clocks) and $|\gamma|$ is the number of interactions. Finally, t shows the total verification time required for GI^h invariant computation and satisfiability checking of $GI^h \wedge \neg\Psi$ and t_{yices} specifies the satisfiablity

checking time required by the Sat-solver. Henceforth, we use the same notation for the tables relative to the other checked systems.

RTD-Finder succeeds in verifying the P_1 property in few minutes for 500 trains. The components invariants of the the train components are computed only once, since there is a unique BIP *Train* type. This local invariant is later instantiated with the replicated trains numbers. Besides, the computation time relative to the history clocks inequalities is polynomial on the number of components.

We note that thanks to the use of reduction techniques applicable for this particular type of systems, UPPAAL was able to check this property for 124 trains in 2 seconds (we used the UPPAAL 4.0.13 release).

2. (P_2) Mutual exclusion property: The gate is not at *up* location when the train is in the crossing

$$P_2 \equiv (\exists j.at(In_j) \wedge \bigwedge_{i \neq j} at(Far_i)) \Rightarrow \neg at(up)$$

The counterexample-based invariant refinement algorithm was necessary for verifying this safety property. The verification results are illustrated in Table 9.2.

Trains number	Property	n	$\|\gamma\|$	d_{max}	p	k_{cex}	t_{cex}
3	P_2	5	8	22	440	1	$0.6s$
5	P_2	7	12	22	2452	1	$3.2s$
10	P_2	12	22	22	22982	1	$45s$
20	P_2	22	42	22	199192	1	$19m45s$

Table 9.2: Verification results for P_2 property

For a spurious counterexample, d is the length of the path from the suspected state (l^θ, ζ^θ) to the symbolic states set $\mathcal{P}$ having an empty preimage. Intuitively, the depth d is the number of backward computation steps required to deduce the invalidity of the counterexample. The number d_{max} is the maximum depth d among the analyzed spurious counterexamples. By p, we note the total number of all the symbolic states computed and visited during the backward analysis and contained in the $\mathcal{P}$ sets, while by k_{cex} we refer to the number of analyzed counterexamples. The total verification time is t_{cex}.

3. Deadlock-freedom

The deadlock freedom property was verified considering that the TGC system is parameterized. Following the small model theorem explained in Chapter 6, it suffices to check the property for all numbers of trains ranging from 1 to 5. The total RTD-Finder verification time for those *small models* is 1.4*s*.
This unifrom verification method serves also to check the P_1 property since it has the form of a T-assertion and the global invariant implies it.

9.2 Token Ring System

The protocol depicted in Figure 9.2 is an adaption from [Rei12] where each process is linked to a timer component. We introduce instead timed processes. The token ring system consists of n processes numbered P^1 to P^n. They are organized in a unidirectional *ring*. A process possesses the *token* if it is not in a^i location. It should stay within this location for at least 2 time units, imposed by the guard of s^i transition. In the token ring protocol, every process P^i receives the token from P^{i-1} through the interaction (s^{i-1}, r^i). It then moves to t_1^i location and after passing the token, it moves from t_2^i to a^i, meaning that it possesses the token. Once P^i delivers the token to the next process, it cannot have it again before 2 time units. This constraint is expressed on the clock x^i. Initially, P^1 is in t_1^1 location while all the other processes P^i are at location a^i, waiting for the reception of the token. The property that we want to check states that exactly one process is not at a^i location at a given time.

$$\Psi = \exists i. \forall j \neq i. \big(\neg at(a^i) \wedge at(a^j)\big)$$

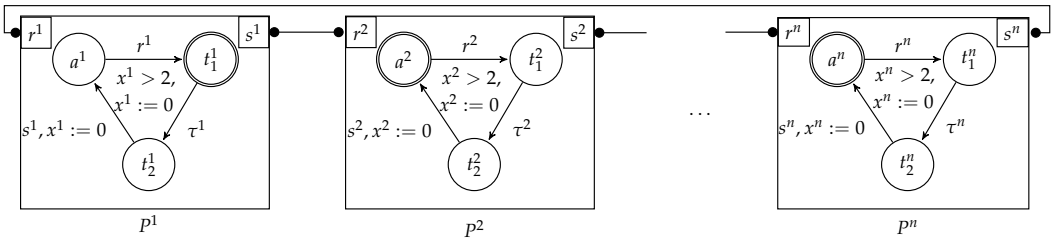


Figure 9.2: A timed token ring system

The property has the form of a T-assertion, which means that the small model theorem presented in Chapter 7 can be applied. What is interesting about this system is that the interaction invariant implies that the token is not lost while the conjunction of the other invariants implies that at most one process possesses the

ring at any given time. Formally, the interaction invariant is expressed as follows:

$$\exists j.\,(at(t_1^j) \vee at(t_2^j))$$

All in all, the global invariant implies the desired property: one and only one process possesses the ring. The global invariant can be expressed as a T-assertion containing 3 universal quantifies. It follows from the results in Chapter 7 that it suffices to verify the system from token ring systems ranging from 1 to 5 processes. The overall verification time required by RTD-Finder to check these small models is equal to $0.4\,s$. We recall that this time is sufficient to show the validity of the property for any number of processes in the token ring protocol.

9.3 Temperature Control System

This benchmark is a timed adaptation of the temperature control (TC) system in [BBSN08]. It represents a simplified model of a nuclear plant. The system consists of a controller interacting with an arbitrary number of rods aimed to maintain the temperature within some bounds (between 450 and 900). When the reactor spends 900 units of time in heating, a rod must be used to cool the reactor.

We verified two properties:

1. (P_3) At least one rode is ready to take *cool* action together with the controller when necessary (deadlock-freedom) in this state. For one rod, $\mathcal{E}(\gamma)$ is enough to show the property. However, for more rods, since the interactions are conflicting, the separation constraints are needed and they bring in new invariants as $\wedge_i(h_{rest_i} - h_{rest_j} \geq 1350)$ for any two different indices i and j.

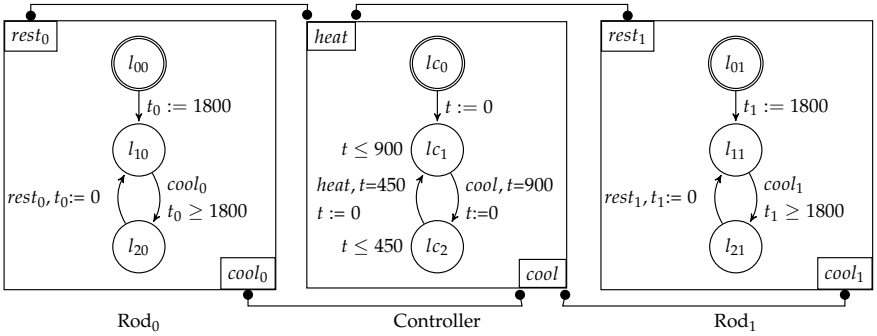


Figure 9.3: A Controller interacting with two rods

Formally, the property P_3 is expressed as follows.

$$P_3 \equiv at(lc_1) \wedge \bigwedge_i at(l_{1i}) \Rightarrow \exists j.(t_j - t \geq 900)$$

The generated invariant is tight enough to imply the property P_3 and the verification results are shown in Table 9.3.

We made a comparison of RTD-Finder with UPPAAL on this system. We increased the number of rods and compared the verification time. For 10 rods, UPPAAL generated no results after five hours and 436519 explored states. Nevertheless, RTD-Finder checked the property for 300 rods in few minutes, as shown in Table 9.3.

Rods number	Property	n	q	c	$\|\gamma\|$	h	t	t_{yices}
20	P_3	21	42	21	40	42	$0.07s$	$0.01s$
50	P_3	51	102	51	100	102	$0.35s$	$0.04s$
100	P_3	101	204	102	200	204	$3.7s$	$0.08s$
300	P_3	301	602	302	600	602	$5m47s$	$0.9s$

Table 9.3: Verification results for P_3 property

2. (P_4) No rod is in cool position if the controller and the remaining rods are in heat position. For a given rod Rod_i, the following property should be satisfied.

$$P_4 \equiv (at(lc_1) \wedge \bigwedge_{j \neq i} at(l_{1j})) \Rightarrow at(l_{1i})$$

The counterexample analysis module is needed for P_4 property. The verifi-

Rods number	Property	n	$\|\gamma\|$	d_{max}	p	k_{cex}	t_{cex}
3	P_4	4	6	7	48	1	$0.2s$
5	P_4	6	10	7	218	1	$0.6s$
20	P_4	21	40	7	8914	1	$3m46s$
50	P_4	51	100	7	128794	1	$14h14m$

Table 9.4: Verification results for P_4 property

cation time is visibly less important when the invariant is strong enough to detect the desired property. In some cases, even when the counterexample analysis is needed, RTD-Finder remains competitive to model checking using forward reachability analysis. This is for instance the case of the temperature control system and (P_4) property which is verified in 3 minutes for 20 rods using the counterexample analysis module compared to the inability to check the property in 5 hours for 10 rods with UPPAAL.

9.4 Gear Controller System

The third benchmark is taken from [LPY98] and models a gear controller system embedded inside vehicles. A gear controller is composed of an interface sending gear change requests to a gear controller component which interacts with the engine, the clutch and the gear-box components. The interface sends signals to the controller to change the gear. The controller interacts with the engine, the clutch and the gear-box. The engine is responsible for regulating the torque or synchronizing the speed while the gear-box sets the gear between some fixed bounds. The clutch works as the gear-box and it is used whenever the engine is not able to function correctly (for example in case of difficult driving conditions). The components are illustrated in Appendix B. One requirement that such a system should satisfy in order to be correct is *predictability*. We verified the following properties after making abstraction from the data variables of the system:

1. *Predictability*:
   - (P_5) When the engine is regulating the torque, the clutch should be closed.
   $$P_5 \equiv at(Torque) \Rightarrow at(Closed)$$
   - (P_6) When the engine is regulating torque, the gear has to be set in the gear-box.
   $$P_6 \equiv (\bigwedge_{i=1...5} at(Gear) \wedge at(Gear_i)) \Rightarrow at(Torque)$$
2. Error detection: The controller detects and indicates the precise errors when the clutch is not opened or closed at time and when the gear-box is unable to set or release a gear at time.
   $$P_7 \equiv at(CCloseError) \Rightarrow at(ErrorClose)$$
   $$P_8 \equiv at(CCloseError) \Rightarrow at(ErrorClose)$$
   $$P_9 \equiv at(GSetError) \Rightarrow at(ErrorIdle)$$
   $$P_{10} \equiv at(GNeuError) \Rightarrow at(ErrorNeu)$$
3. The gear controller system is deadlock-free.

RTD-Finder verified the predictability properties (P_5) and (P_6) as well as the error detection properties (from P_7 to P_{10}) in 15.2 seconds while the deadlock-freedom was checked in 17.6 seconds. The global invariant was tight enough to check all of them.

9.5 Dual Chamber Implantable Pacemaker

We considered the verification of a dual chamber implantable pacemaker modeled and verified in [JPM+12]. Its illustration is given in Figure 9.4. The system is designed to manage the cardiac rhythm. In the considered pacemaker mode, both the atrium and ventricle of the heart are paced. Based on the sensing of both chambers, the pacing can be restrained or activated. For a safe operation, it is essential that the ventricles of the heart should not be paced beyond a maximum rate equal to a $TURI$ constant. A ventricle pace (VP) can occur at least TURI time units after a ventricle event. This requirement expresses the *Upper rate limit* property.

We summarize the verified properties in the following:

1. (P_{11}) There is a minimum time elapse $TURI$ between a ventricle (VS) sense and a ventricle pace (VP) event. This LTL property is verified by adding the matching monitor component to the system and synchronizing its actions (VP and VS) to the other components through interactions. We verified that when the monitor reaches the location *interval*, its clock t is greater than $TURI$. The corresponding property is $at(interval) \rightarrow t \geq TURI$.

   As in [JPM+12], we verified both of the properties by translating them into a monitor component. The interval between a *VS venticular event* and a *VP venticular event* should be longer than $TURI$. In addition, our method offers another way to check the first property without resorting to the monitor. We expressed it by means of the already introduced history clocks. The difference between the interactions history clocks relative to those two events is bigger than the desired time elapse.

2. (P_{12}) There is a minimum time elapse $TURI$ between two ventricle pace (VP) events. This property is verified by use of a monitor. However, using history clocks to express the safety requirement is not possible for this second property since it compares two occurrences of the same action VP.

3. The pacemaker system is deadlock-free.

RTD-Finder verified (P_{11}) property in 0.25 seconds and the computed invariant was sufficient for its detection while (P_{12}) and the deadlock-freedom where verified in 0.6 seconds and 0.5 seconds respectively. The counterexample-based invariant refinement module was needed to eliminate 28 and 11 spurious counterexamples respectively.

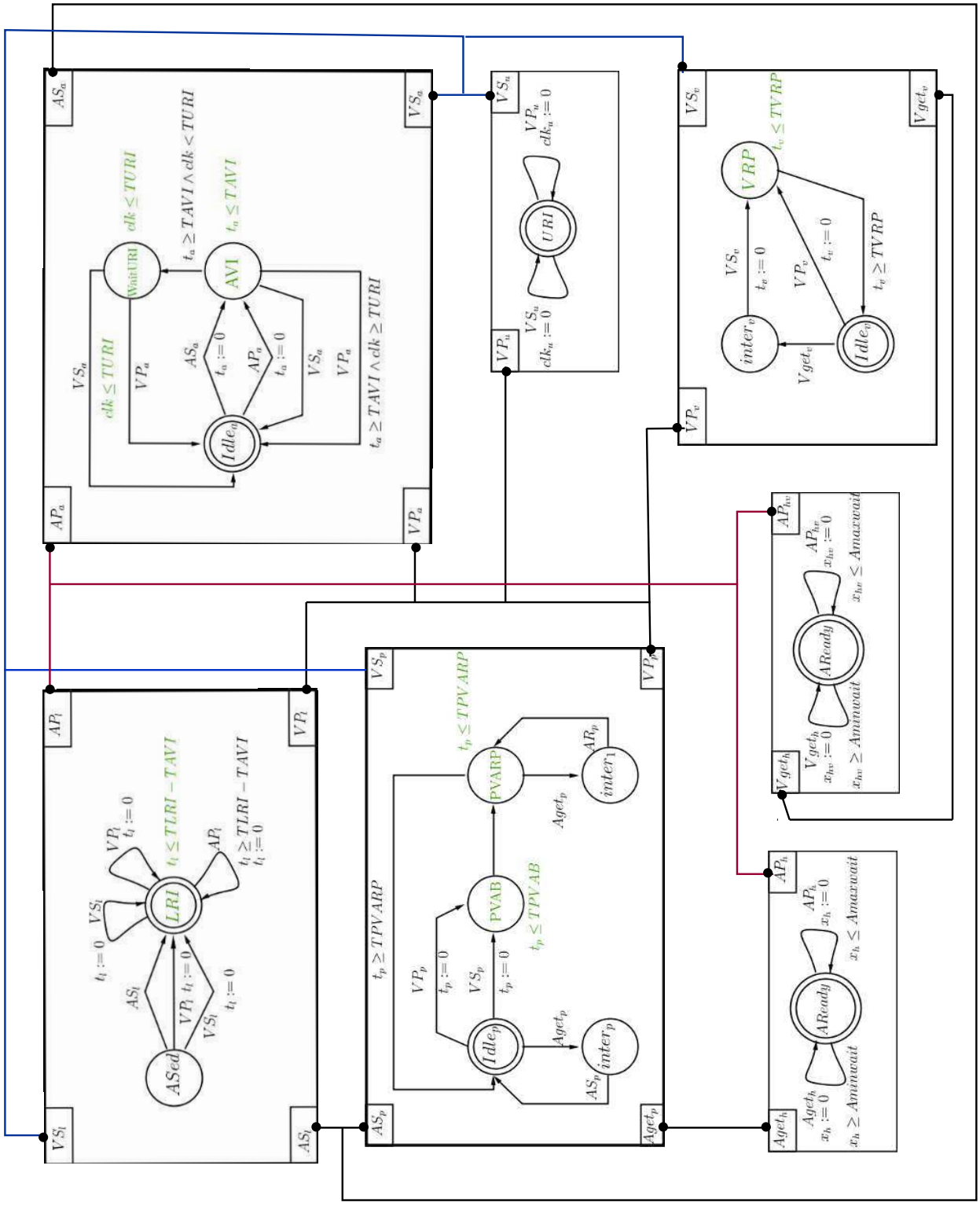


Figure 9.4: Pacemaker system

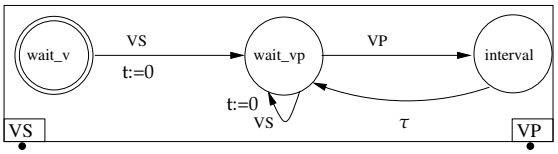


Figure 9.5: Monitor for the upper rate limit property

9.6 Summary

This chapter provided the experimental results for the verification of various case studies using RTD-Finder. They have shown the efficiency of the proposed invariant computation method for many properties. When the global invariant is strong enough and when no counterexample analysis is needed, RTD-Finder offers generally a drastic reduction of the verification time in comparison with UPPAAL. In some cases, it succeeds in verifying systems composed of hundreds of components while UPPAAL is unable to exceed 20 of them.
We mention also the important reduction of the verification time for the previously introduced class of parameterized timed systems. For this particular class, the validity of the property is asserted for all numbers of replicas by relying on the verification of the small models, which is usually drastically rapid.

Chapter 10

Conclusions and Perspectives

In this chapter, we conclude the thesis by recalling the main objectives of the work, the achieved goals and the future work directions.

10.1 Conclusions

The main objective of the present work is to contribute to resolving the combinatorial explosion problem inherent to the verification of real-time systems. At this aim, we proposed a fully compositional and automatic method for timed invariant generation designated for safety properties verification. It is based on the deductive approach and a verification rule that states that if an over-approximation of the reachable states set implies the property of interest, then the system satisfies it. In this work, one main contribution consists in answering to the question: how to compute such an over-approximation for timed systems in a fully compositional manner?
In the context of real-time systems, the question is even hardened due to the synchronous model of time. In fact, while the clocks of the different components advance at the same rate, the local analysis of each component is performed in isolation with respect to the others. The use of the history clocks lays on the core of our answer. The local invariants of the components extended with the actions history clocks contain relations between them and the original local clocks. In conjunction with the invariants relating the different history clocks, they induce relations between the clocks of the different components. In our framework, components are modeled as timed automata and are glued together by a set of

multiparty interactions. This modeling framework is constructive and is aimed to subserve compositional verification methods: the properties of components are preserved with parallel composition and the global properties can be inferred from the local properties of the components.
The method is implemented in the RTD-Finder tool where the input system is modeled in the RT-BIP language. As shown in the experiments section, where a comparison with the UPPAAL tool is conducted, the global invariant was strong enough to capture the safety property of interest in many cases. There, the verification time is drastically reduced in comparison with exploration-based techniques, particularly when the number of components is very large. Beyond its scalability, the proposed approach has other advantages.

- Unlike model-checking where the verification of generalizations is not possible, it allows the uniform verification of parameterized timed systems. In fact, the global invariant can be expressed at a particular form suitable for the application of a small model theorem.
- In case a valid counterexample is detected, the correction of the global system does not necessarily go through its full reconstruction.

10.2 Perspectives

Following the above contributions, several extensions and improvements are sought to enhance the scalability of the method or enable covering additional aspects and hence ensure more generality and applicability.

Method Extension

Counterexamples Analysis Module Improvement At the current version of the tool, the counterexamples are analyzed by simple backward reachability computation. We proposed to restrict the preimage computation to satisfy the global invariant at each step. At the implementation level, this requires specific data structures to efficiently restrict and manipulate the sets of symbolic states. Furthermore, a rich set of reduction techniques that have been widely studied in the literature may be implemented to reduce the complexity of this backward computation task.

Handling Richer Classes of Models

Urgency types

Two different approaches may be considered to restrict a component, or more generally a system, from remaining infinitely at a given location. While in this work, we focus on time-progress-conditions, the urgency framework [BS00] is proposed at the aim of ensuring additionally a well-timedness property of systems, which is *reactivity*, meaning that if no discrete transition can fire, then time can progress in the current location of the component. Besides the guard, a transition is labeled with a deadline. Their relative position defines the degree of urgency: lazy, delayable and eager transitions. One future work direction would be a method for compositional invariant generation for such systems. One major difficulty for a direct application of our method is that in the urgency framework, the computation of the local invariants based on the reachability graph would result rather in weak component invariants. In fact, due to the urgency types, the actual time progress on locations depends from the interactions and is not only locally determined.

Parametric Timed Systems

The method can be extended to parametric timed systems, that is timed systems containing unknown constraints. The aim would be inferring parameter valuations consistent with given safety properties. To reason parametrically, unknown time delays can be perceived as parameters. Parametric timed automata [AHV93] were proposed to approach this problem. Several verification and synthesis techniques were proposed for parametric timed automata. We cite the works in [HRSV01], [AS11] and [JLR13]. In relation with our framework, the use of parametric DBM would allow to generate parameteric invariants for components modeled by parameteric timed automata.
Based on the work in [CARB15], a parametric invariant of the system may serve as a cheap over-approximation of the reachable states set allowing to formulate a $\exists\forall$SMT satisfiability problem for parameter synthesis.

Properties

Deadlock

RTD-Finder offers high scalability especially when the property to check is not combinatorial. Checking the absence of deadlock is currently more problematic. If in the untimed case one can provide an exact formalization of deadlock by means of local characterizations, this is no longer the case for timed systems. More precisely,

in a timed context, the condition which expresses that an interaction is eventually enabled cannot be decided by the consideration of the involved components only, but depends also from the timing constraints of the non involved components. The use of approximation techniques could be envisaged to counter this problem.

Timelock

Unlike timed systems where the notion of progress is linked to discrete state changes, progress in timed systems may result from the passing of time. A state of the system contains a timelock if for every path starting from this state, time can advance only to a certain value, while by nature time always progresses. A system is not timelock-free if at least one of its states has a timelock. The verification of such a property for real-time systems is of high interest.

Temporal properties

Currently, RTD-Finder handles only safety properties but we want to study the extension to check Temporal Logic Properties. Linear Temporal Logic (LTL) properties by use of Timed Buchi automata is possible. Actually, LTL properties [Pnu77] can be transformed into state-based safety properties. This translation lays on the construction of a deterministic progress monitor component representing the LTL property of interest. An adaptation of the bounded LTL synthesis method in [FS13] for the timed setting is detailed in [CARB15].
With the rich research advancement in timed models, some timed extensions of LTL logic were proposed. We mention the Metric temporal Logic (MTL) [Koy90] where the modalities are augmented with timing constraints. To extend our method to such properties, their transformation to state-based safety properties could be analogously studied.

Further applications of the Global Invariant

Beyond the verification of safety properties, the compositional invariant generation method can be applied for other purposes. For instance, we mention the runtime verification and the distributed implementation of real-time systems.

Distributed implementations of real-time systems One solution to face communication delays in distributed systems consists in notifying the participating components in advance. However, due to the synchronous model of time, the scheduler should be able to observe, not only the components participating in the scheduled interaction, but also the non participating components such that their

time-progress-conditions are not violated at the execution time. In [Tri15][TCB15], a solution was proposed at the aim of minimizing the number of observed components. The authors formulate a predicate characterizing if for the scheduled interaction, a given component can be correctly removed from the set of components that should be necessarily observed. In [Tri15][TCB15], RTD-Finder is used as a static analyzer to check the satisfiability of such a predicate. This work is being generalized to remove some restriction on the systems, for instance the non determinism condition.

In a general manner, the invariants can be used in the distributed setting in order to infer information about the other components based on local knowledge, hence help partial decision making.

Bibliography

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138:3–34, 1995. (Cited on pages 29 and 105.)

[ACKS02] Gilles Audemard, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani. Bounded model checking for timed systems. In *Formal Techniques for Networked and Distributed Systems - FORTE 2002, Proceedings*, pages 243–259, 2002. (Cited on page 8.)

[ACS13] Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Rigorous implementation of real-time systems - from theory to application. *Mathematical Structures in Computer Science*, 2013. (Cited on pages 12 and 122.)

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 1994. (Cited on pages 42 and 47.)

[ADM05] Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Closed, open, and robust timed networks. *Electronic Notes in Theoretical Computer Science*, 138(3):117–151, 2005. (Cited on page 104.)

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999. (Cited on page 18.)

[AH96] Rajeev Alur and Thomas A. Henzinger. Reactive modules. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 207–218, 1996. (Cited on page 16.)

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601, 1993. (Cited on page 151.)

[AJ99] Parosh Aziz Abdulla and Bengt Jonsson. On the existence of network invariants for verifying parameterized systems. In *Correct System Design*, pages 180–197. 1999. (Cited on page 104.)

[AJ03] Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003. (Cited on page 104.)

[AJNd02] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Regular model checking made simple and effcient. In *International Conference on Concurrency Theory*, pages 116–131. Springer, 2002. (Cited on page 104.)

[AK86] Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986. (Cited on page 104.)

[AL95] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Trans. Program. Lang. Syst.*, 17(3):507–534, 1995. (Cited on page 16.)

[AMN05] Rajeev Alur, P. Madhusudan, and Wonhong Nam. Symbolic compositional verification by learning assumptions. In *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, pages 548–562, 2005. (Cited on page 17.)

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. (Cited on page 17.)

[AS11] Étienne André and Romain Soulat. Synthesis of timing parameters satisfying safety properties. In *Reachability Problems - 5th International Workshop, RP 2011, Genoa, Italy, September 28-30, 2011. Proceedings*, pages 31–44, 2011. (Cited on page 151.)

[BBBL13] S. Bensalem, M. Bozga, B. Boyer, and A. Legay. Incremental generation of linear invariants for component-based systems. In *Application of Concurrency to System Design , 13th International Conference, ACSD*, pages 80–89, 2013. (Cited on pages 33 and 36.)

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim Guldstrand Larsen. Static guard analysis in timed automata verification. In *TACAS, Warsaw, Poland, April 7-11, 2003, Proceedings*, pages 254–277, 2003. (Cited on page 7.)

[BBLP06] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *STTT*, 8(3):204–215, 2006. (Cited on page 7.)

[BBNS09] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. D-finder: A tool for compositional deadlock detection and verification. In *Proceedings of Computer Aided Verification, 21st International Conference, CAV*, pages 614–619, 2009. (Cited on pages 10, 11 and 34.)

[BBSN08] Saddek Bensalem, Marius Bozga, Joseph Sifakis, and Thanh-Hung Nguyen. Compositional verification for component-based systems and application. In *Proceedings of Automated Technology for Verification and Analysis, 6th International Symposium, ATVA*, pages 64–79, 2008. (Cited on pages 10, 33, 34 and 143.)

[BCC+02] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software,f invited chapter. In T. Mogensen, D.A. Schmidt, and I.H. Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566, pages 85–108. Springer Verlag, 2002. (Cited on page 3.)

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS*, pages 193–207, 1999. (Cited on page 8.)

[BCH+05] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H Roux. Comparison of the expressiveness of timed automata and time petri nets. In *Formal Modeling and Analysis of Timed Systems*, pages 211–225. Springer, 2005. (Cited on page 42.)

[BCP07] Albert Benveniste, Benoît Caillaud, and Roberto Passerone. A generic model of contracts for embedded systems. *CoRR*, abs/0706.1456, 2007. (Cited on page 19.)

[BDH01] Dragan Bosnacki, Dennis Dams, and Leszek Holenderski. A heuristic for symmetry reductions with scalarsets. In *FME: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Proceedings*, pages 518–533, 2001. (Cited on page 6.)

[BDH+12] Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Moving from specifications to contracts in component-based design. In *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 43–58, 2012. (Cited on page 20.)

[BDL+06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems QEST*, pages 125–126, 2006. (Cited on pages 7 and 139.)

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In *Computer Aided Verification, 10th International Conference, CAV, Proceedings*, pages 546–550, 1998. (Cited on page 7.)

[BFM+08] Luca Benvenuti, Alberto Ferrari, Leonardo Mangeruca, Emanuele Mazzi, Roberto Passerone, and Christos Sofronis. A contract-based formalism for the specification of heterogeneous systems (invited). In *Forum on specification and Design Languages, FDL 2008, September 23-25, 2008, Stuttgart, Germany, Proceedings*, pages 142–147, 2008. (Cited on page 19.)

[BGL+11] Saddek Bensalem, Andreas Griesmayer, Axel Legay, Thanh-Hung Nguyen, and Doron Peled. Efficient deadlock detection for concurrent systems. In *MEMOCODE*, pages 119–129, 2011. (Cited on page 100.)

[BGP03] Howard Barringer, Dimitra Giannakopoulou, and Corina S Pasareanu. Proof rules for automated compositional verification through learning. In *Workshop on Specification and Verification of Component Based Systems, Proceedings*, 2003. (Cited on page 17.)

[BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *CONCUR Concurrency Theory, 9th International Conference, Proceedings*, pages 485–500, 1998. (Cited on page 5.)

[BJNT00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *Computer Aided Verification*, pages 403–418, 2000. (Cited on page 104.)

[BLPR09] Nathalie Bertrand, Axel Legay, Sophie Pinchinat, and Jean-Baptiste Raclet. A compositional approach on modal specifications for timed systems. In *Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM, Proceedings*, pages 679–697, 2009. (Cited on page 20.)

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time petri nets. In *Proceedings IFIP*, 1983. (Cited on page 42.)

[BMMR01] Thomas Ball, Rupak Majumdar, Todd Millstein, and Sriram K Rajamani. Automatic predicate abstraction of c programs. In *ACM SIGPLAN Notices*, volume 36, pages 203–213. ACM, 2001. (Cited on page 3.)

[Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Form. Methods Syst. Des.*, 2004. (Cited on page 51.)

[BPG08] Mihaela Gheorghiu Bobaru, Corina S. Pasareanu, and Dimitra Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, pages 135–148, 2008. (Cited on page 18.)

[BR70] J. N. Buxton and B. Randell, editors. *Software Engineering Techniques: Report of a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27-31 Oct. 1969, Brussels, Scientific Affairs Division, NATO*. 1970. (Cited on page 2.)

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. (Cited on page 6.)

[BS98] Sébastien Bornot and Joseph Sifakis. An algebraic framework for urgency. *Information and Computation*, 1998. (Cited on page 48.)

[BS00] Sébastien Bornot and Joseph Sifakis. An algebraic framework for urgency. *Inf. Comput.*, 163(1):172–202, 2000. (Cited on page 151.)

[BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Proceedings of Lectures on Concurrency and Petri Nets*, 2003. (Cited on pages 51 and 124.)

[CAC08] Jamieson M. Cobleigh, George S. Avrunin, and Lori A. Clarke. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Trans. Softw. Eng. Methodol.*, 17(2), 2008. (Cited on page 19.)

[CARB15] Chih-Hong Cheng, Lacramioara Astefanoaei, Souha Ben Rayana, and Saddek Bensalem. Timed orchestration for component-based systems. *CoRR*, abs/1504.05513, 2015. (Cited on pages 151 and 152.)

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977. (Cited on page 3.)

[CFH+03] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003. (Cited on page 6.)

[CGJ+00] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, CAV, 2000. (Cited on page 6.)

[CGP03] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Proceedings*, pages 331–346, 2003. (Cited on pages 17 and 18.)

[CGR10] Alessandro Carioni, Silvio Ghilardi, and Silvio Ranise. MCMT in the land of parametrized timed automata. In *6th International Verification Workshop, VERIFY*, pages 47–64, 2010. (Cited on page 104.)

[Cla08] Edmund M. Clarke. 25 years of model checking. 2008. (Cited on page 89.)

[CLM89] Edmund M. Clarke, David E. Long, and Kenneth L. McMillan. Compositional model checking. In *Proceedings of the Fourth Annual Symposium*

on Logic in Computer Science (LICS '89), pages 353–362, 1989. (Cited on page 16.)

[CM89] K. Mani Chandy and Jayadev Misra. *Parallel program design - a foundation*. Addison-Wesley, 1989. (Cited on page 16.)

[Cou91] Costas Courcoubetis. Minimum and maximum delay problems in real-time systems. In *Computer Aided Verification, 3rd International Workshop, CAV, Aalborg, Denmark, July, 1-4, 1991, Proceedings*, pages 399–409, 1991. (Cited on page 75.)

[CW96] Edmund M Clarke and Jeannette M Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996. (Cited on page 4.)

[dAH01a] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001*, pages 109–120, 2001. (Cited on page 19.)

[dAH01b] Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In *Embedded Software, First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October, 8-10, 2001, Proceedings*, pages 148–165, 2001. (Cited on page 19.)

[dAHS02] Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Timed interfaces. In *Embedded Software, Second International Conference, EMSOFT 2002, Grenoble, France, October 7-9, 2002, Proceedings*, pages 108–122, 2002. (Cited on page 20.)

[DdM06] Bruno Dutertre and Leonardo de Moura. The Yices SMT solver. Technical report, SRI International, 2006. (Cited on page 123.)

[DHLP04] Alexandre David, John Håkansson, Kim G. Larsen, and Paul Pettersson. Minimal dbm substraction. In *Nordic Workshop on Programming Theory*, 2004. (Cited on pages 46 and 93.)

[DKL07] Henning Dierks, Sebastian Kupferschmid, and Kim G. Larsen. Automatic abstraction refinement for timed automata. In *FORMATS*, 2007. (Cited on page 7.)

[DLL+10] Alexandre David, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. ECDAR: an environment for compositional

design and analysis of real time systems. In *Automated Technology for Verification and Analysis - 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010. Proceedings*, pages 365–370, 2010. (Cited on page 20.)

[Dou98] Bruce Powel Douglass. *Real-Time UML*. Addison-Wesley, Reading, 1998. (Cited on page 42.)

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS '98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, pages 313–329, 1998. (Cited on page 7.)

[Dut14] Bruno Dutertre. Yices 2.2. In *Proceedings of Computer Aided Verification - 26th International Conference, CAV*, pages 737–744, 2014. (Cited on page 123.)

[DY96] Conrado Daws and Sergio Yovine. Reducing the number of clock variables of timed automata. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96), December 4-6, 1996, Washington, DC, USA*, pages 73–81, 1996. (Cited on page 7.)

[FCJK05] Ansgar Fehnker, Edmund Clarke, Sumit Jha, and Bruce Krogh. Refining abstractions of hybrid systems using counterexample fragments. In *In Hybrid Systems: Computation and Control, volume 3414 of Lect. Notes Comput. Sci*, pages 242–257, 2005. (Cited on page 6.)

[FHD+99] Thomas Firley, Michaela Huhn, Karsten Diethers, Thomas Gehrke, and Ursula Goltz. Timed sequence diagrams and tool-based analysis: A case study. In *Proceedings of the 2Nd International Conference on The Unified Modeling Language: Beyond the Standard, UML*, pages 645–660, 1999. (Cited on page 42.)

[FHK04] Goran Frehse, Zhi Han, and Bruce Krogh. Assume-guarantee reasoning for hybrid i/o-automata by over-approximation of continuous interaction. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 479–484. IEEE, 2004. (Cited on page 19.)

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–357, june 1962. (Cited on page 47.)

[FS13] Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *STTT*, 15(5-6):519–539, 2013. (Cited on page 152.)

[GJL04] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Learning of event-recording automata. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, pages 379–396, 2004. (Cited on page 18.)

[GJL10] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Learning of event-recording automata. *Theor. Comput. Sci.*, 411(47):4029–4054, 2010. (Cited on page 18.)

[GJP06] Olga Grinchtein, Bengt Jonsson, and Paul Pettersson. Inference of event-recording automata using timed decision trees. In *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, pages 435–449, 2006. (Cited on page 18.)

[GL91] Orna Grumberg and David E. Long. Model checking and modular verification. In *CONCUR '91, 2nd International Conference on Concurrency Theory, , Proceedings*, pages 250–265, 1991. (Cited on page 16.)

[GL08] Olga Grinchtein and Martin Leucker. Network invariants for real-time systems. *Formal Aspects of Computing*, 20(6):619–635, 2008. (Cited on page 104.)

[GNRZ08] Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, and Daniele Zucchelli. Towards SMT model checking of array-based systems. In *4th International Joint Conference on Automated Reasoning, IJCAR*, pages 67–82, 2008. (Cited on page 104.)

[God90] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *Computer Aided Verification, 2nd International Workshop, CAV, Proceedings*, pages 176–185, 1990. (Cited on page 5.)

[God96] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996. (Cited on page 5.)

[GPB05] Dimitra Giannakopoulou, Corina S. Pasareanu, and Howard Barringer. Component verification with automatically generated assumptions. *Autom. Softw. Eng.*, 12(3):297–320, 2005. (Cited on page 17.)

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987. (Cited on page 42.)

[HBL+03] Martijn Hendriks, Gerd Behrmann, Kim Guldstrand Larsen, Peter Niebert, and Frits W. Vaandrager. Adding symmetry reduction to uppaal. In *Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS*, pages 46–59, 2003. (Cited on page 6.)

[HCM+08] John Håkansson, Jan Carlson, Aurelien Monot, Paul Pettersson, and Davor Slutej. Component-based design and analysis of embedded systems with UPPAAL PORT. In *Automated Technology for Verification and Analysis, 6th International Symposium, ATVA, Proceedings*, pages 252–257, 2008. (Cited on page 5.)

[Hen02] Martijn Hendriks. *Enhancing uppaal by exploiting symmetry*. Nijmegen Institute for Computing and Information Sciences, University of Nijmegen, 2002. (Cited on page 6.)

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57:94–124, 1998. (Cited on page 105.)

[HKSLT02] Stefan Haar, Laurent Kaiser, Françoise Simonot-Lion, and Joël Toussaint. Equivalence of timed state machines and safe tpn. In *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pages 119–124. IEEE, 2002. (Cited on page 42.)

[HMU03] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003. (Cited on page 83.)

[Hol97] Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5):279–295, 1997. (Cited on page 6.)

[HP94] Gerard J. Holzmann and Doron A. Peled. An improvement in formal verification. In *Formal Description Techniques VII, Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques*, pages 197–211, 1994. (Cited on page 5.)

[HRSV01] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. In *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European*

Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings, pages 189–203, 2001. (Cited on page 151.)

[ID96] C. Norris Ip and David L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2):41–75, 1996. (Cited on page 6.)

[JLR13] Aleksandra Jovanovic, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for timed automata. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS, Rome, Italy, March 16-24, 2013. Proceedings*, pages 401–415, 2013. (Cited on page 151.)

[JM94] F. Jahanian and A.K. Mok. Modechart: a specification language for real-time systems. *Software Engineering, IEEE Transactions on*, 20(12):933–947, 1994. (Cited on page 42.)

[JM06] Ranjit Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS, 2006. (Cited on page 6.)

[JM12] Taylor T Johnson and Sayan Mitra. A small model theorem for rectangular hybrid automata networks. In *FMOODS/FORTE*, pages 18–34, 2012. (Cited on pages 104, 105, 106 and 107.)

[JN00] Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 220–235. 2000. (Cited on page 104.)

[Jon83] Cliff B. Jones. Tentative steps toward a development method for interfering programs. *ACM Trans. Program. Lang. Syst.*, 5(4):596–619, 1983. (Cited on page 16.)

[JPM+12] Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *TACAS*, 2012. (Cited on page 146.)

[KMR02] Alexander Knapp, Stephan Merz, and Christopher Rauh. Model checking - timed uml state machines and collaborations. In *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: Co-sponsored by IFIP WG 2.2*, FTRTFT '02, pages 395–416, 2002. (Cited on page 42.)

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. (Cited on page 152.)

[KP07] Stephanie Kemper and André Platzer. Sat-based abstraction refinement for real-time systems. *Electron. Notes Theor. Comput. Sci.*, 182:107–122, 2007. (Cited on page 7.)

[LAL+14] S. W. Lin, E. André, Y. Liu, J. Sun, and J. S. Dong. Learning assumptions for compositionalverification of timed systems. *IEEE Transactions on Software Engineering*, 40(2):137–153, Feb 2014. (Cited on page 18.)

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977. (Cited on page 16.)

[LHR01] David Lesens, Nicolas Halbwachs, and Pascal Raymond. Automatic verification of parameterized networks of processes. *Theoretical Computer Science*, 256:113–144, 2001. (Cited on page 104.)

[LL98] François Laroussinie and Kim G Larsen. Cmc: A tool for compositional model-checking of real-time systems. In *Formal Description Techniques and Protocol Specification, Testing and Verification*, pages 439–456. 1998. (Cited on page 7.)

[LNW06] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Interface input/output automata. In *FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings*, pages 82–97, 2006. (Cited on page 20.)

[LPWY99] Kim Guldstrand Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nord. J. Comput.*, 6(3):271–298, 1999. (Cited on page 45.)

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997. (Cited on page 7.)

[LPY98] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal design and analysis of a gear controller. In *TACAS*, 1998. (Cited on pages 145 and 179.)

[LSV03] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, 2003. (Cited on page 19.)

[MC81] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Trans. Software Eng.*, 7(4):417–426, 1981. (Cited on page 16.)

[McM93] Kenneth L McMillan. *Symbolic model checking*. Springer, 1993. (Cited on page 6.)

[McM97] Kenneth L. McMillan. A compositional rule for hardware design refinement. In *Computer Aided Verification, 9th International Conference, CAV '97, Proceedings*, pages 24–35, 1997. (Cited on page 16.)

[Mey92] Bertrand Meyer. Applying "design by contract". *IEEE Computer*, 25(10):40–51, 1992. (Cited on page 19.)

[Mil89] Robin Milner. *Communication and concurrency*, volume 84. Prentice hall New York etc., 1989. (Cited on page 4.)

[Min99] Marius Minea. *Partial order reduction for verification of timed systems, thesis*. PhD thesis, Citeseer, 1999. (Cited on page 5.)

[MP95] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems - safety, a book*. Springer, 1995. (Cited on page 9.)

[Ngu10] Thanh-Hung Nguyen. *Constructive Verification for Component-based Systems, a thesis report*. Theses, Institut National Polytechnique de Grenoble - INPG, May 2010. (Cited on pages 38 and 123.)

[NMA08] Wonhong Nam, P. Madhusudan, and Rajeev Alur. Automatic symbolic compositional verification by learning assumptions. *Formal Methods in System Design*, 32(3):207–234, 2008. (Cited on page 17.)

[OG76] Susan S. Owicki and David Gries. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM*, 19(5):279–285, 1976. (Cited on page 16.)

[OGO06] Iulian Ober, Susanne Graf, and Ileana Ober. Validating timed uml models by simulation and verification. *International Journal on Software Tools for Technology Transfer*, 8(2):128–145, 2006. (Cited on page 42.)

[PDMV13] Pavithra Prabhakar, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Hybrid automata-based cegar for rectangular hybrid systems. In *Verification, Model Checking, and Abstract Interpretation*, 2013. (Cited on page 7.)

[Pel93] Doron A. Peled. All from one, one for all: on model checking using representatives. In *Computer Aided Verification, CAV, 1993, Proceedings*, pages 409–423, 1993. (Cited on page 5.)

[Pel94] Doron A. Peled. Combining partial order reductions with on-the-fly model-checking. In *Computer Aided Verification, 6th International Conference, CAV, Proceedings*, pages 377–390, 1994. (Cited on page 5.)

[PGB+08] Corina S. Pasareanu, Dimitra Giannakopoulou, Mihaela Gheorghiu Bobaru, Jamieson M. Cobleigh, and Howard Barringer. Learning to divide and conquer: applying the l* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008. (Cited on page 18.)

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57, 1977. (Cited on page 152.)

[Pnu85] A. Pnueli. Logics and models of concurrent systems. chapter In Transition from Global to Modular Temporal Reasoning About Programs, pages 123–144. Springer-Verlag New York, Inc., 1985. (Cited on page 16.)

[PRZ01] Amir Pnueli, Sitvanit Ruah, and Lenore D. Zuck. Automatic deductive verification with invisible invariants. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 82–97, 2001. (Cited on page 104.)

[QG08] Sophie Quinton and Susanne Graf. Contract-based verification of hierarchical systems of components. In *Sixth IEEE International Conference on Software Engineering and Formal Methods, SEFM 2008, Cape Town, South Africa, 10-14 November 2008*, pages 377–381, 2008. (Cited on page 20.)

[Ram74] Chander Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. 1974. (Cited on page 42.)

[Rei12] Johannes Reich. Processes, roles and their interactions. In *Proceedings of IWIGP*, 2012. (Cited on page 142.)

[Rep96] Inquiry Board Report. Ariane 5: Flight 501 failure. Technical report, The European Space Agency, 1996. (Cited on page 1.)

[RS93] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993. (Cited on page 17.)

[Srb08] Jiří Srba. Comparing the expressiveness of timed automata and timed extensions of petri nets. In *Formal Modeling and Analysis of Timed Systems*, pages 15–32. Springer, 2008. (Cited on page 42.)

[Sta85] Eugene W. Stark. A proof technique for rely/guarantee properties. In *Foundations of Software Technology and Theoretical Computer Science, Proceedings*, pages 369–391, 1985. (Cited on page 16.)

[TCB15] Ahlem Triki, Jacques Combaz, and Saddek Bensalem. Optimized distributed implementation of timed component-based systems. In *13. ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE*, pages 30–35, 2015. (Cited on page 153.)

[Tri15] Ahlem Triki. *Distributed Implementations of Timed Component-based Systems*. PhD thesis, Grenoble Alpes, 2015. (Cited on page 153.)

[TWS06] Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. Real-time interfaces for composing real-time systems. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software, EMSOFT*, pages 34–43, 2006. (Cited on page 20.)

[TY01] Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001. (Cited on page 6.)

[Val89] Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Proceedings]*, pages 491–515, 1989. (Cited on page 5.)

[VDWW06] Sicco E Verwer, Mathijs M De Weerdt, and Cees Witteveen. Identifying an automaton model for timed data. In *Proceedings of the 15th Annual Machine Learning Conference of Belgium and the Netherlands, Ghent, Belgium, 11-12 May 2006*, 2006. (Cited on page 18.)

[VHB+03] Willem Visser, Klaus Havelund, Guillaume Brat, SeungJoon Park, and Flavio Lerda. Model checking programs. *Automated Software Engineering*, 10(2):203–232, 2003. (Cited on page 3.)

[Wan00] Farn Wang. Efficient data structure for fully symbolic verification of real-time software systems. In *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS, Proceedings*, pages 157–171, 2000. (Cited on page 6.)

[WL90] Pierre Wolper and Vinciane Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, pages 68–80, 1990. (Cited on page 104.)

[WS02] Farn Wang and Karsten Schmidt. Symmetric symbolic safety-analysis of concurrent software with pointer data structures. In *Formal Techniques for Networked and Distributed Systems - FORTE, Proceedings*, pages 50–64, 2002. (Cited on page 6.)

[WT95] Howard Wong-Toi. *Symbolic approximations for verifying real-time systems*. PhD thesis, stanford university, 1995. (Cited on page 7.)

[YW99] Masahiro Yamauchi and Toshimasa Watanabe. Time complexity analysis of the minimal siphon extraction problem of petri nets. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 82(11):2558–2565, 1999. (Cited on page 37.)

Appendices

Appendix A

RT-BIP Syntax

A.1 BIP Tool-Chain

BIP offers a practical implementation of the three layers of BIP introduced at Chapter 3 as follows:

- The Behavior is specified by a set of *components*.
- The *connectors* serve to structure the interactions. A connector corresponds to a subset of interactions.
- *Priorities* serve to resolve conflicts between interactions.

Concretely, the main constructs of the BIP language are:

- atomic component: to model the behavior described by a set of transitions labeled by port names.
  The ports serve for interface to interact with the other components, taking the role of actions as defined in Chapter 4. In some cases, the interface is enriched with data variables visible to the other components.
- connector: to model the synchronization between the ports of different components.
- priority: to restrict among the possible interactions based on the states of the interacting components.
- coumpound components: to compose a set of atomic components or other coumpound components.
  The term *component* refers to either an atomic component or a coumpound

component. Both types of components have ports and optionally data variables.

- model: to encapsulate the definition of the different components types, and to describe the top level instance of the system to simulate, analyze or verify.

Compared to the untimed BIP framework, the main difference of RT-BIP lies on the behavior layer which is extended to timed automata (and timed Petri nets). In the following, we show the different elements of RT-BIP models.

In RT-BIP, the different types for components, connectors and ports are first defined. These types are later instantiated, meaning that a given type may be instantiated many times. For example, if many atomic components share the same behavior, it is sufficient to model their common behavior in one atomic component type and instantiate it later as many times as required.

The system illustrated in Figure 4.5 is composed of a controller interacting with two workers. This Appendix shows how this system is modeled in the RT-BIP language.
The two components $Worker_0$ and $Worker_1$ are perceived as two different instantiations of a same atomic type *Worker*. Its description in RT-BIP language is illustrated in the following:

```
port type Port()


atom type Worker()
  clock y
  export port  Port d()
  export port  Port b()

  place l1, l2

  initial to l1

  on b
    from l1 to l2
    when (y >= 8 second)
    do {}

  on d
    from l2 to l1
    reset y
    do {}
```

```
end
```

The definition of the `Worker` atomic type is preceded by the definition of a port type `Port`. In the definition of `Worker` atomic component type, two locations are defined, which are l_1 and l_2. Since the ports `d` and `b` are *exported*, they are intended to interact with ports of other components. The clock y is reset whenever the transition labeled with d is executed, moving the component from location l_2 to l_1 whereas the transition labeled with b has for guard $(y \geq 8)$ and moves the component from l_1 to l_2 location. The constructor *initialto* precises which location is initial, and eventually some possible initial update function to execute. Each transition is labeled with a port (after **on**), a source location and a destination location. The update functions and the guards on clocks are have a C-like syntax.

The definition of the `Controller` atomic type contains the definition of a time progress condition `tpc1`. The time progress conditions on locations are specified by the `invariant` key. The time progress condition `tpc1` is linked to location lc_1 and introduces the constraint $x \leq 4$ *second*, meaning that the clock x cannot exceed 4 seconds at location lc_1.

```
atom type Controller()
  clock x

  export port  Port a()
  export port  Port c()
  port Port initi()

  place lc0, lc1, lc2

  initial to lc0

  on  initi
    from lc0 to lc1
    when (x >= 8 second)
    reset x
    do {}

  on a
    from lc1 to lc2
    when (x == 4 second)
    reset x
    do {}

  on c
```

```
    from lc2 to lc1
    reset x
    do {}

   invariant tpc1 at lc1  when (x<= 4 second)

end
```

The port `initi` is not exported, it is not designed to interact with ports of other ports and is related to the internal transition from lc_0 to lc_1.

A connector type is parameterized by a list of ports describing its support. The construct `define` gives the types of the associated ports. For each interaction, a guard and a data transfer function can be eventually defined. In the following definition, the `Link2` connector models a strong synchronization between two ports of type `Port` from two different components. The guard associated to the interaction is `true` by default.

```
connector type Link2(Port p1, Port p2)
  define  p1 p2
end
```

A compound component is defined by gathering instances of predefined atomic components or other compound types. The components instances are glued together through instantiated connectors synchronizing instantiated ports.

```
compound type CW()

  component Worker worker1()
  component Worker worker2()
  component Controller controller()

  connector Link ab1(worker1.b, controller.a)
  connector Link ab2(worker2.b, controller.a)
  connector Link cd1(worker1.d, controller.c)
  connector Link cd2(worker2.d, controller.c)

end
```

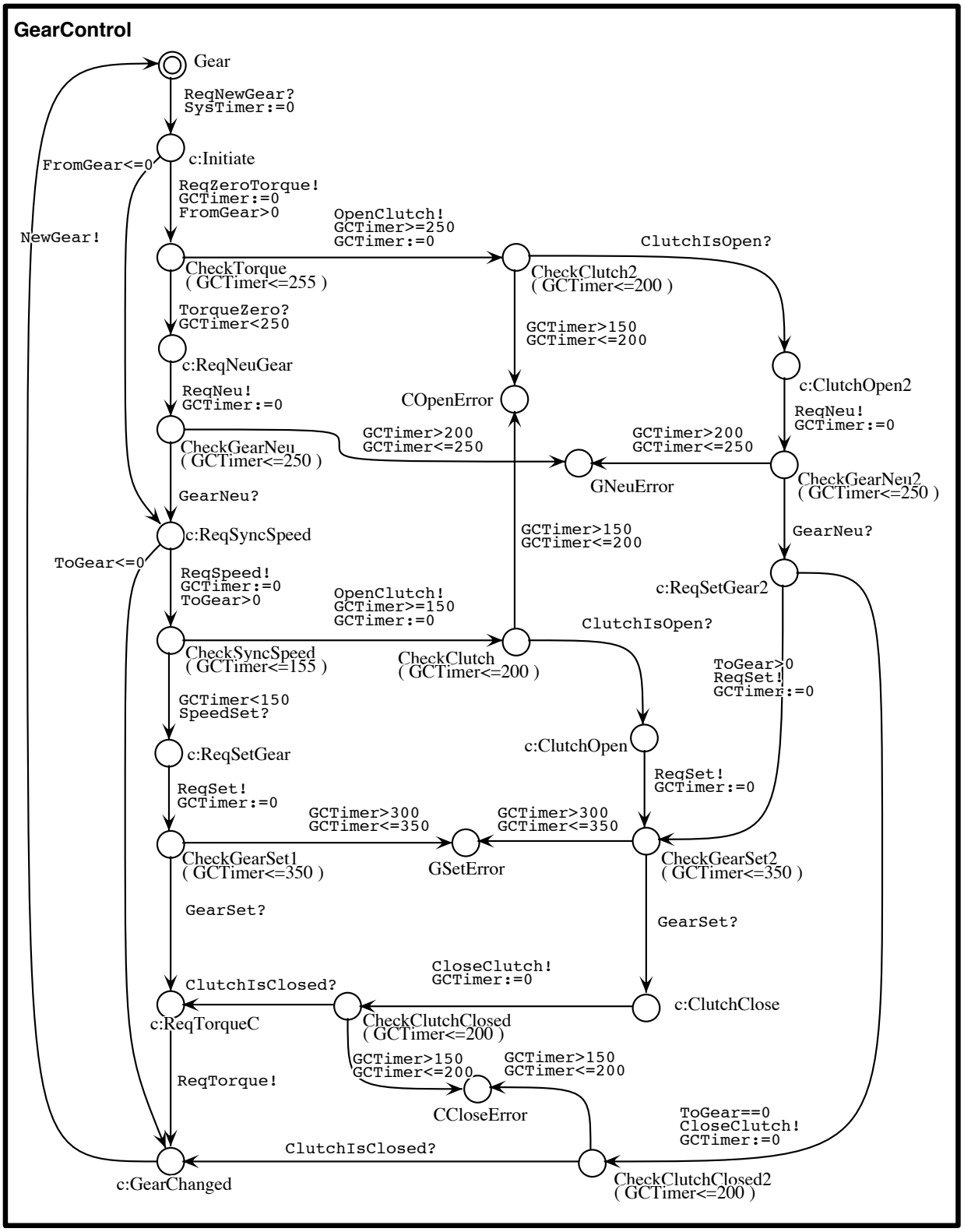
During the instantiation of the main compound, the three components that constitute the system, `worker1`, `worker2` and `controller`, are first instantiated. This induces the instantiation of the respective ports. For example, for `worker1`, the port instances `worker1.b` and `worker1.d` are created, referring to ports *b* and *d* of the atomic component *worker1*.

Afterwards, the instantiation of the connectors is performed. For instance, the connector `ab1` links port `b` of `worker1` with port `a` of `controller`.

Appendix B

Gear Controller System

The gear controller system is composed of the GearControl, Interface, GearBox, Clutch and Engine components, depicted respectively in figures B.1, B.5, B.2, B.3 and , B.4. This figures are taken from [LPY98] and illustrate the components as in UPPAAL tool. Translating the system to a BIP model, all the interactions between the components are binary. The GearControl component interacts with Interface through one interaction synchronizing their **NewGear** actions and one interaction synchronizing their **NewGear** actions. The Interface component keeps record of whether GearControl is changing gear or idling. GearControl performs the gear change in 5 steps. After reception of the gear change request from Interface, a zero torque transmission is performed, meaning that the current gear can be changed. Once the gear is released, the GearControl applies the synchronous speed mode over the transmission and sets a new gear. The engine torque is subsequently increased.
In case of difficulty in driving, the engine may become unable to perform zero torque or synchronous speed over transmission, in which case the gear change is accomplished my means of the clutch. By opening it, the connection between the wheels and the engine is broken and the gear is able to set and release the new gear. A complete and detailed presentation of the functionality and the error states of the system can be found in [LPY98].


GearControl
Gear
ReqNewGear?
SysTimer:=0
c:Initiate
FromGear<=0
ReqZeroTorque!
GCTimer:=0
FromGear>0
OpenClutch!
GCTimer>=250
GCTimer:=0
NewGear!
ClutchIsOpen?
CheckTorque
(GCTimer<=255)
CheckClutch2
(GCTimer<=200)
TorqueZero?
GCTimer<250
GCTimer>150
GCTimer<=200
c:ReqNeuGear
c:ClutchOpen2
ReqNeu!
GCTimer:=0
COpenError
ReqNeu!
GCTimer:=0
GCTimer>200
GCTimer<=250
GCTimer>200
GCTimer<=250
CheckGearNeu
(GCTimer<=250)
GNeuError
CheckGearNeu2
(GCTimer<=250)
GearNeu?
GearNeu?
c:ReqSyncSpeed
GCTimer>150
GCTimer<=200
ToGear<=0
ReqSpeed!
GCTimer:=0
ToGear>0
OpenClutch!
GCTimer>=150
GCTimer:=0
c:ReqSetGear2
ClutchIsOpen?
CheckSyncSpeed
(GCTimer<=155)
CheckClutch
(GCTimer<=200)
ToGear>0
ReqSet!
GCTimer:=0
GCTimer<150
SpeedSet?
c:ClutchOpen
c:ReqSetGear
ReqSet!
GCTimer:=0
ReqSet!
GCTimer:=0
GCTimer>300
GCTimer<=350
GCTimer>300
GCTimer<=350
CheckGearSet1
(GCTimer<=350)
GSetError
CheckGearSet2
(GCTimer<=350)
GearSet?
GearSet?
CloseClutch!
GCTimer:=0
ClutchIsClosed?
c:ClutchClose
c:ReqTorqueC
CheckClutchClosed
(GCTimer<=200)
GCTimer>150
GCTimer<=200
GCTimer>150
GCTimer<=200
ReqTorque!
CCloseError
ToGear==0
CloseClutch!
GCTimer:=0
ClutchIsClosed?
c:GearChanged
CheckClutchClosed2
(GCTimer<=200)

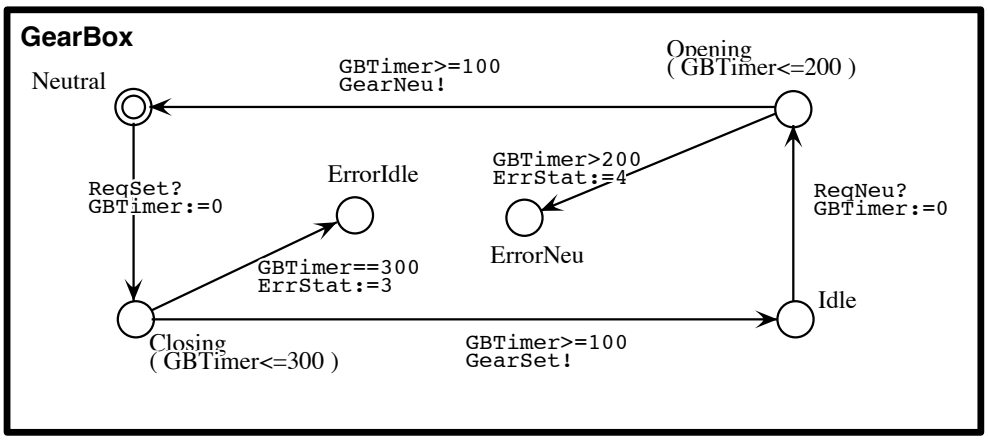


Figure B.2: The GearBox component

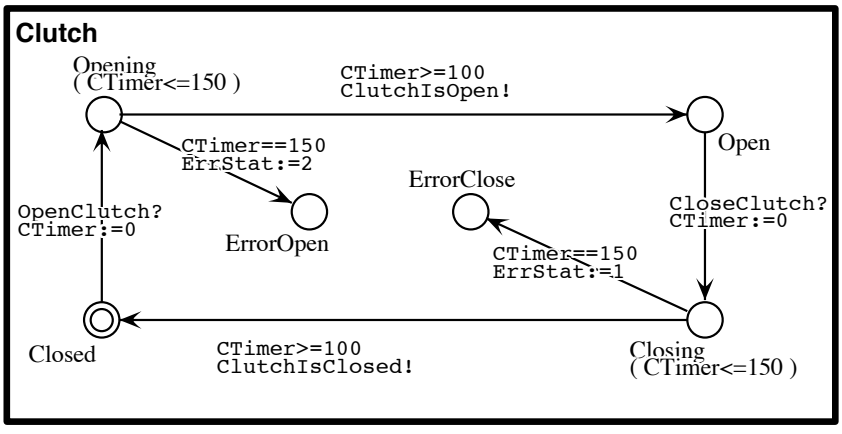


Figure B.3: The Clutch component

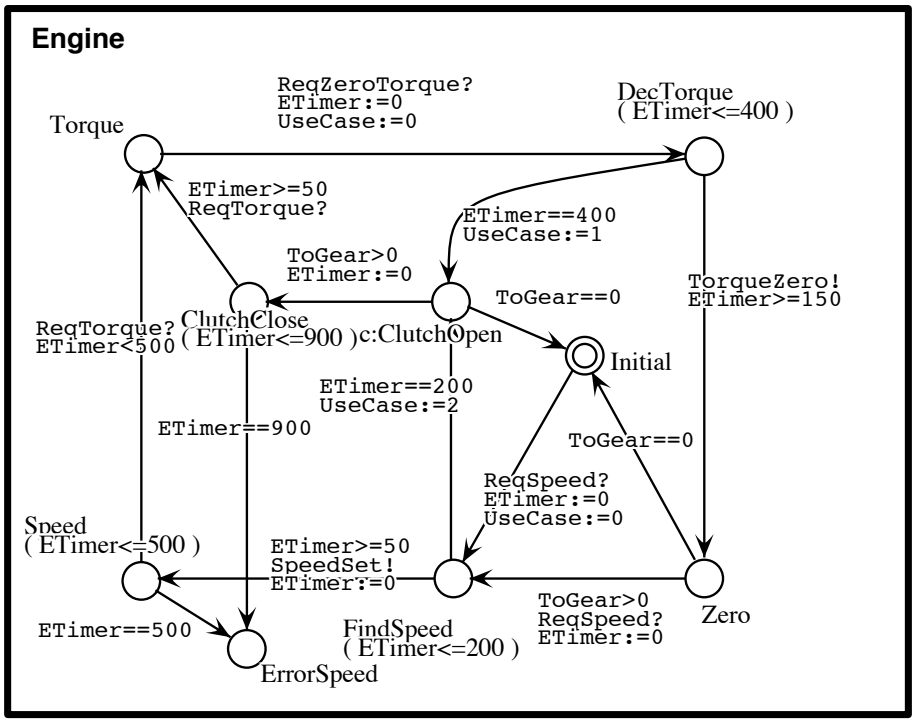


Figure B.4: The Engine component

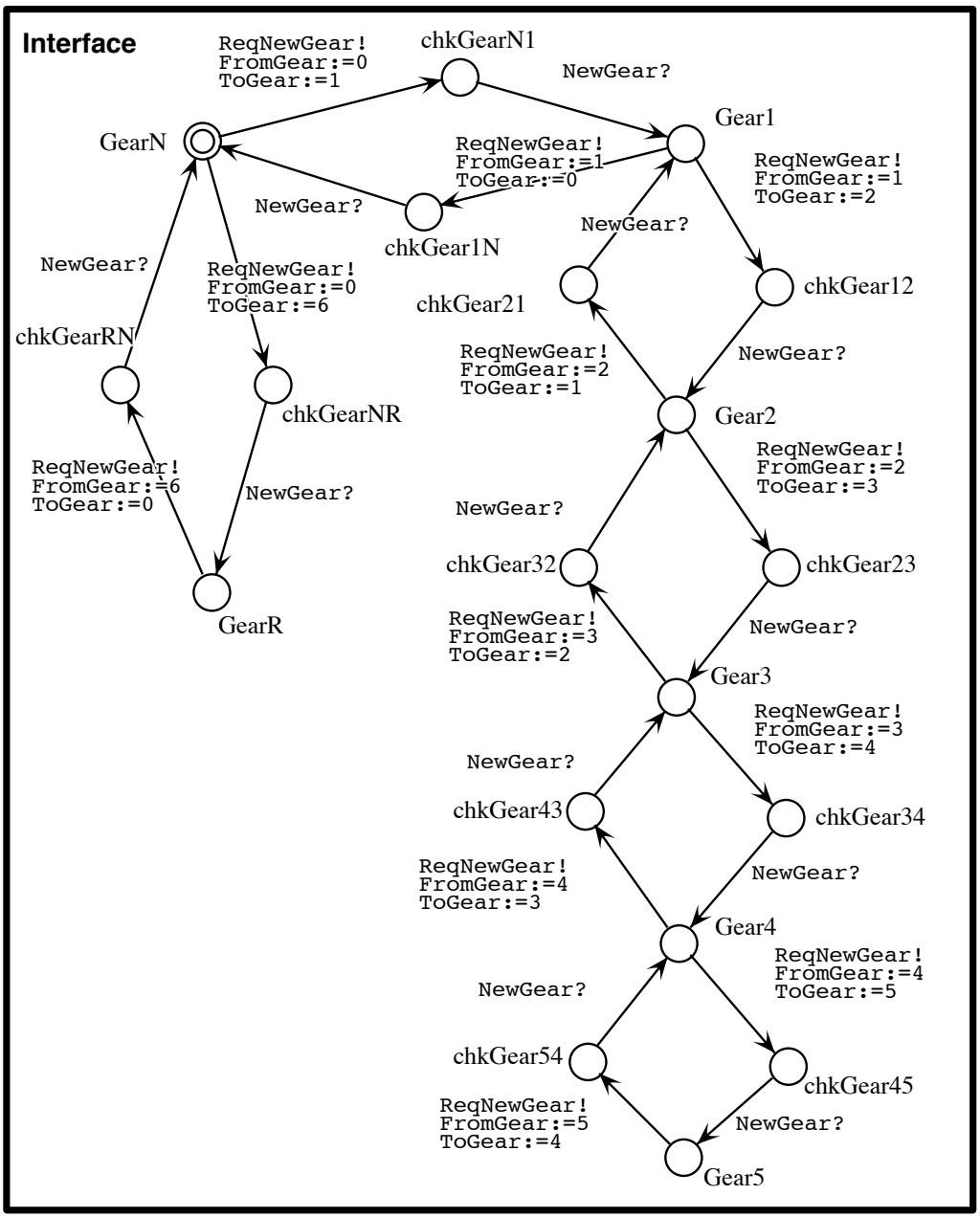


Figure B.5: The Interface component

Appendix C

Related Publications

Journal Paper

1. Souha Ben Rayana, Lacramioara Astefanoaei, Saddek Bensalem, Marius Bozga, Jacques Combaz: **Compositional Verification for Timed Systems Based on Automatic Invariant Generation**. In *Logical Methods in Computer Science (LMCS)*, 11(3) (2015)

International Conferences

2. Souha Ben Rayana, Marius Bozga, Saddek Bensalem, Jacques Combaz: **RTD-Finder: A Tool for Compositional Verification of Real-Time Component-Based Systems**. In *TACAS 2016*, pages 394-406

3. Lacramioara Astefanoaei, Souha Ben Rayana, Saddek Bensalem, Marius Bozga, Jacques Combaz: **Compositional Verification of Parameterised Timed Systems.** In *NFM 2015*, pages 66-81

4. Lacramioara Astefanoaei, Souha Ben Rayana, Saddek Bensalem, Marius Bozga, Jacques Combaz: **Compositional Invariant Generation for Timed Systems.** In *TACAS 2014*, pages 263-278