UNIVERSITÀ DEGLI STUDI DI TRENTO

INTERNATIONAL DOCTORATE SCHOOL IN INFORMATION AND COMMUNICATION TECHNOLOGIES

XXIII CYCLE – 2012

Parametric Real-Time System Feasibility Analysis Using Parametric Timed Automata

Yusi Ramadian

UNIVERSITÀ DEGLI STUDI DI TRENTO

INTERNATIONAL DOCTORATE SCHOOL IN INFORMATION AND COMMUNICATION TECHNOLOGIES

XXIII CICLO - 2012

Yusi Ramadian

Parametric Real-Time System Feasibility Analysis Using Parametric Timed Automata

Luigi Palopoli	(Advisor)
Alessandro Cimatti	(Co-Advisor)

Thesis Committee

Marco Di Natale
Gianluca Dini

AUTHOR'S ADDRESS:
Yusi Ramadian
Dipartimento di Ingineeria e Scienza dell'Informazione
Università degli Studi di Trento
via Sommarive 14, I-38050 Povo di Trento, Italy
E-MAIL: `yusi.ramadian@disi.unitn.it`
WWW: `http://disi.unitn.it/users/yusi.ramadian`

Abstract

Real-time applications are playing an increasingly significant role in our life. The cost and risk involved in their design leads to the need for a correct and robust modelling of the system before its deployment. Many approaches have been proposed to verify the schedulability of real-time task system. A frequent limitation is that they force the task activation to restrictive patterns (e.g. periodic). Furthermore, the type of analysis carried out by the real-time scheduling theory relies on restrictive assumptions that could make the designers miss important optimization opportunities. On the other hand, the application of formal methods for verification of timed systems typically produces a yes/no answer that does not suggest any correction action or robustness margins of a given design.

This work proposes an approach to combine the benefits of formal method in terms of flexibility with the production of a clear feedback for the designers. The key idea is to use parametric timed automata to enable the definition of flexible task activation patterns. The Parametric Verification of Temporal Properties (PTVP) algorithm proposed in this work produces a region of feasible parameters for real-time system. All the parameter valuation within this region is guaranteed to make the system respect the desired temporal behaviour. In this way developers are provided with a richer information than the simple feasibility of a given design choice.

This method uses symbolic model checking technique to produce the result that is a union of polyhedral regions in the parameter space associated with feasible parameters. It is implemented in the tool Quinq that is based on NuSMV3. The tool also implemented an optimization to speed up the search, such as using non-parametric model checker to find counterexamples (i.e. traces) related to the unfeasible choices of parameters.

Two applications of the tool and of the underlying method to several real-time system examples are presented in this dissertation : periodic real-time system tasks with offset and heterogeneous distributed real-time systems. A work that applies the tool in collaboration with another real-time system analysis tool, Modular Performance Analysis Toolbox, is also presented to show one of the many possible application of the method presented in this work.

In this work we also compare our approach to the state of the art in the field of sensitivity analysis of real-time systems. However, compared to the other tools and approaches in this field, the method offered in this work presents unique advantages in the generality of the system modelling approach and the possibility to analyse the entire region of feasibility of any desired parameter in the system.

Keywords

Formal method, real-time system, temporal verification, scheduling verification.

Acknowledgments

Bismillah hir Rahman nir Rahim, Alhamdulillah hi Rabbil Alameen, Praise be to Allah Almighty who always provide me with everything more than I have ever asked to bring this PhD research and dissertation into completion.

For this PhD research I am deeply indebted to my supervisors, Luigi Palopoli and Alessandro Cimatti, who always have their faith in me and my thesis work and always been very supportive and helpful. They always provided me with fresh ideas of new directions and knowledge I needed for every obstacles I found throughout the research. I also appreciate all the moral supports they provided which play important role for me to conclude this work.

During the implementation I got significant supports by everyone in NuSMV developer team of Fondazione Bruno Kessler. Sergio Mover in particular has provided significant helps towards the hybrid extension of the work in the end. MathSAT developer, Alberto Griggio, has contributed his help through MathSAT tool since the beginning phase of this work up to the end with the additional existelim tool. I would also like to thank Alena Simalatsar and Roberto Passerone for the collaboration work we had.

All my love and forever gratitude to my beloved parents, Dra. Psi. Nurdiah Tedjowati and Farid Luthfi, M.M, who had given me never ending care, encouragement and support all the time that has enabled me to reach this point in my life. For my siblings, Reza Rahardian, Faddy Ardian, Denia Farahdian. I would also like to thank very much my dearest husband, Tizar Rizano, for his support all along this PhD program. I could not be anything I am now without them.

Last, I hope that the work in this thesis will be of value and be my contribution for the knowledge in general.

Trento, April 24th 2012
Yusi Ramadian

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Description

Real-time applications are increasingly popular especially in industrial control, and in the automotive domain. A significant example is offered by Anti-Lock Brake System in a car (ABS). In this case, a computation activity (henceforth called *task*) triggered by an event has to release the brake within a certain time limit before the wheel is locked by the braking action. Therefore, the application has to ensure that its function is carried out within its deadline. Applications like this are now commonplace and play important roles in our everyday life.

Designing a real-time application is to be regarded as a challenging activity. Indeed, not only has the programmer has to take care of the correctness of the project, but he/she has also to make sure that the application will meet its timing requirements and generally speaking will comply with its non functional requirements.

Traditional approaches to real-time applications design were based on extensive prototyping activities. An evident drawback is that the costs may be unaffordable, while the safety standard remain very low (a sufficient coverage of all possible situations is very difficult to achieve). More importantly, if a system malfunctioning is detected, the developer is provided with small or no feedback information at all to change the design. Therefore, these approaches generate over-commitment of the application producers with respect to a specific hardware vendor. Indeed, evaluating different hardware choices by hand-crafting prototypes is far too expensive. After a very demanding *tailoring* has been done on a specific hardware, the management develops a natural reluctance toward porting the whole package to a different platform.

To overcome the limitations of these *blind-folded* approaches there is a widespread agreement on facing problems at the highest possible level of abstraction. In the early phases of the development the developer should be able to: 1)identify potential timing faults *before* prototyping the system, 2) evaluate different hardware and software alternatives, 3) have a clear assessment of possible actions to be taken in response to a problem (e.g., simplifying some computations to reduce their resource requirements). In the recent years there have been different proposals for design methodologies that go along this direction. It is worth citing the so-called platform based approaches [17, 48]. The core of these methodologies is the adoption of Computer Aided Design (CAD) tools that aid the developer with such activities as code generation, system level simulation and timing analysis.

In particular, the work presented in this dissertation deals with timing analysis based on verification tools. In this field, there has been an extensive research work that produced both academic results and industrial tools. Reference [54] gives a complete overview of the state of the art in this domain. However, most of the available tools suffer two major limitations :

1. Strong assumptions on the activation patterns of the real-time activities: some of these tools consider worst case activation patterns (e.g., periodically spaced out activations) that maximize the system workload, but could be overly conservative for the specific problem at hand.

2. Provides no or limited feedback to the developer: most of the available verifiers decide whether a set of real-time tasks meets its deadlines. The result only applies to a specific choice of the design parameters. If the developer changes something in the design, a new verification process is required. Conversely, if the result of the verification is negative, the developer does not receive any feedback on the parameters to be changed.

   On recent tools, given a specific choice of the design parameter as reference input values, some sensitivity analysis feedback can be performed, resulting into the computation of the tolerance (slacks) in which some parameter can change while preserving the validity of certain temporal properties. However, these tools heavily relying on the initial input value provided by the user or consider very specific conservative situations.

In contrast, the method proposed in this thesis is able to handle flexible activation patterns without making restrictive assumptions. Moreover, it provides the information of the entire region in the parameter space for which the verification yields negative or positive results without relying on any reference input value. The possible uses of this information are very many. For instance, if a specific implementation belongs to the region of feasible parameters, then it is possible to gauge the robustness of the solution (e.g., how much can one vary the parameters without compromising the feasibility). On the contrary, if a design is evaluated as infeasible the designer knows which parameters should be changed. Finally, we can explore alternative solutions for the hardware implementation that achieve feasibility.

1.2 Solution Overview

In this dissertation, we advocate the use of Parametric Timed Automata (PTA) to model real–time systems. This approach allows us to represent a parametric real–time system retaining a large level of freedom on the specification of the activation and of the interaction patterns of the different tasks in the system. In this setting, the violation of temporal properties is encoded as a reachability problem of one or more undesired locations. For this problem, we develop an algorithmic solution, the Parametric Verification of Temporal Properties (PTVP) algorithm, that used a symbolic model checker to derive linear constraints for the task parameters that solve the reachability problem. The obtained linear constraints are then used to construct the infeasible region of parameter space and subsequently provide us with the feasibility region of the system. This idea is implemented in a tool, Quinq, that is based on the NuSMV [22] model checker. The tool offers a variety of possible options for modelling the system, which we display in their application to a set of meaningful case studies.

1.3 Research Contribution

The contributions we made during our PhD research are the following:

- Establishing parametric timed automata as a powerful and promising means to represent parametric real–time system

- Setting up a methodology, based on PTVP algorithm, which enables an effective computation of a region of parameters that respect some required properties;
- Implementation of the complete method in the tool Quinq
- Application of the method and using the tool in several case problems : periodic task system [21], heterogeneous system [39], and in collaboration with Modular Performance Analysis Toolbox (MPA) [53].

The application of the methodology defined in this thesis to a set of case studies is a result of the collaboration with different authors. In particular, the application to heterogeneous and distributed real–time systems is a result of the collaboration with Hoa Le and Roberto Passerone. The connection between PTA and MPA is the result of the collaboration with Alena Simalatsar and Roberto Passerone, and MPA developers: Kai Lampka, Simon Perathoner, and Lothar Thiele.

1.4 Structure of the Thesis

This thesis is organised as follows:

Chapter 2 We introduce the basic definitions and terminology that we will use for real–time systems and for their design cycle. We then formalise the problem that we will target during the rest of the thesis. In the interest of clarity, we also introduce some application scenarios that will guide as through the course of the thesis as running examples. Finally, we spell out the most relevant research challenges addressed by the thesis.

Chapter 3 We offer an overview of the background knowledge that we used as the foundation of our work in this PhD dissertation.

Chapter 4 The parametric timed automata (PTA) model that will be the cornerstone of our thesis is introduced here. The analysis method that is used to check the parametric feasibility of a real–time system presented using PTA, our PTVP algorithm, is outlined in its main conceptual components.

Chapter 5 We analyse in detail the general architecture, and the different components required by the PTVP algorithm and by its implementation in the Quinq tool.

Chapter 6 We present two example applications of the tool: analysis of real–time periodic task sets with offsets and analysis of a distributed real–time system with flexible real–time constraints.

Chapter 7: the interaction of the tool with MPA is presented here as a paradigm of possible synergies with existing tools.

Chapter 8 Describes the related work on the verification of real–time systems with a particular focus on the tools and methodologies that carry out some form of parametric analysis.

Chapter 9 We offer our conclusion hinting to possible future developments and extensions of the approach.

Chapter 2

Real–time Design Challenges

2.1 Real–Time System

In a real–time system, the correctness of the results heavily relies on its timing performance. A failure in delivering the result within a specified temporal constraint (deadline) will make it useless in spite of its correctness. The ability for a system to meet its temporal deadlines largely depends on how shared resource are scheduled when more than one task compete for their access. For this reason, we will use the term *schedulability* to refer to the ability for the system to comply with its timing constraints for a given scheduling policy.

Real-time systems are pervasively present in our daily life. The range of their applications extends from the menial system to such critical systems as the anti-lock braking system (ABS) of the automobiles. Despite their pervasiveness, they often go unnoticed: as long as no violation of their required behaviour occurs, we will always have seamless systems in our hands.

The importance of correct real–time system

1. Safety Consideration

   Examples of safety critical real–time systems are the control system of cars (e.g., the fuel injection control, the ABS, the ESP) and of aircrafts, where a control system supervises the landing and take off process. For these systems, it is important that not even a single deadline is missed, as people safety depends on the correctness of the timing behaviour.

2. Manufacturing consideration

   The embedded systems are usually mass-manufactured to reduce the production cost to the minimum. A failure of a system that is only detected after the manufacture completes will cause a massive recall of the product, which determines potentially relevant economic losses. For this reason it has to be guaranteed that the system is *correct by design* before going into production.

Current real–time system are increasingly variable in their behaviour and have to meet changing environmental constraints. Indeed, at run time, there can be many factors from the environment affecting the system. For this reason, some assumptions that have been made at design time might be violated during the system execution. In this case, the system is required to maintain its functional correctness and the timely delivery of the result. In other words, modern applications are required to be *robust* across a wide range of operating conditions. In this context, a correct real–time system design flow is one that guarantees both the correctness and the robustness of its final outcome.

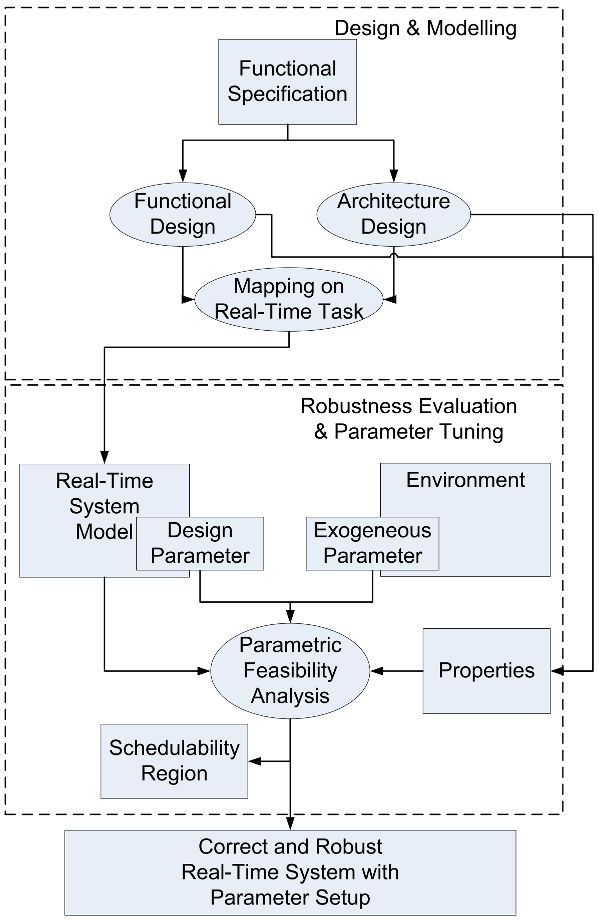


Figure 2.1: The design process of correct design

Designing Real-Time System

The methodology we envisage to achieve the goals discussed above is illustrated in Figure 2.1. The development process could be divided to the following two parts:

1. Designing and modelling

   In designing real–time systems we start from the functional specification of what the system will do. This functional specification then will be implemented through *functional design* and *architectural design*. Functional design specifies subfunctions which, in their interaction, will deliver the functionality. The architectural design specifies the hardware architecture and software infrastructure that will be used for the real–time system. Based on the two designs, the subfunctions will be then mapped to real–time tasks making up the real–time system implementation. The mapped *real–time system model* has to capture the following aspects:

   (a) The activation pattern : How we generate the activation of jobs of the system subfunctions?

   (b) The timing properties : When can we consider a system as correct?

   (c) The scheduling algorithm : How we would schedule the execution of each subfunction when they compete for the same resource?

2. Robustness evaluation and parameter tuning

   After constructing a model of the real–time tasks system, the next step is to asses its schedulability and robustness. In general, the designers need to do the following :

   (a) Assign values for the quantities that fall within the designer control but that are not forced to take a pre-assigned value.
   (b) Evaluate the system robustness with respect to the quantities that are not under control.

   Generally speaking, the model of the system comprises a set of *design parameters*. These variables come from their activation pattern and timing properties of the tasks. For periodic tasks with offsets, design parameters the task periods, the deadline, the computation time and the offset time. Some of these parameters are under our control, some others are already fixed as they are dictated by the choices in the architecture design (e.g., the computation time). From the environment we will also have *exogenous parameters*, that is the external factors that will affect our system during the runtime. For example in a case where an aperiodic task is triggered by an event in the environment, the inter-arrival time between two adjacent events can be considered as an exogenous parameter.

   Generally speaking, the correctness of a real–time system is evaluated on the ground of its ability to respect certain *properties* that derive from its functional and architectural design, first and foremost its schedulability. Therefore, designers are confronted with the problem of assigning the free parameters so that the desired properties are fulfilled and that the robustness of the system against possible variations of the uncontrollable parameters is maximised.

   This leads us to the problem of *Parametric Feasibility Analysis*, which is the main objective of this work. In our view, the system is a multi–dimensional object, where each parameter is associated with a different dimension. In this context, performing Parametric Feasibility Analysis amounts to finding the region in in the multi dimensional space of the parameters such that every point in the region corresponds to an instantiation of the system that respects the required properties. We will refer to this region as the *feasibility region*. This region will support the designer in tuning the parameter towards the desired level of robustness, leading him/her to the design of *a correct and robust real–time system*.

What this thesis is about and what is not

To further clarify what we mean, it is useful to mention several possible queries a designers may raise in the design process. Among the several possible queries one might want to address in this first phase of the design are the following:

- Should single or multiple processors be used for the system?
- What priority should we assign to each tasks?
- Is it be possible to have the system distributed in several devices that then communicate?
- Is it possible to guarantee the integrity of data and bounded delays when tasks share resources?

This type of queries are important to map the functionalities correctly and as efficiently as possible according to the available hardware and software infrastructure. However, they are out of the scope of our thesis. Next, after the mapping of the system, we will need to assess the robustness of the system. We will encounter queries on robustness of the system, such as:

- How much could the computation of a task (or multiple tasks) can be stretched without affecting the system schedulability?
- With a distributed scheme is it possible to guarantee that the communication between the parts will respect certain timing latency criteria?
- Would it be possible for us to add one or several new tasks without affecting the system schedulability?
- How badly would such introduction of new tasks affect the robustness of the system?
- How could we tweak the offset to make the system schedulable?

Queries on robustness like these are the types of queries related to parametric analysis of the system. To answer queries such as *Could we tweak into the offset to make the system schedulable?* for example, we need to introduce the offset of the system tasks as parameters. That way in the parameter analysis of the offset we would be able to see which offset valuation would make the system schedulable. This is the kind of queries that we are interested in.

2.1.1 Terminologies and Definitions

In this section we introduce some basic terminologies and a set of definitions that we will use throughout our discussion.

Task model

A real-time systems $\mathcal{S}$ is in our framework defined as a set of computing activities, henceforth referred to as tasks. A *task* or *process* is an entity of computation executed by an operating system. A real–time system consists of several tasks. For a generic task we will use the symbol τ_i, hence

$$\mathcal{S} = \{\tau_1, \tau_2, \ldots, \tau_n\} .$$

Activation pattern

Each task τ_i is characterised by an *activation pattern*, a certain designed relative deadline D_i, and a worst case execution time C_i. A task τ_i generates a possibly infinite stream of jobs $J_{i,1}, J_{i,2}, \ldots$. Each job $J_{i,1}$ is characterised by [31]:

1. arrival time $a_{i,j}$: the time when the job is activated or released so that it is ready for execution
2. start time $s_{i,j}$: the time when the job is actually started being executed
3. finishing time $f_{i,j}$: the time when the job finishes its execution
4. absolute deadline $d_{i,j} = a_{i,j} + D_i$: the limit time before which the job must have finished its execution.

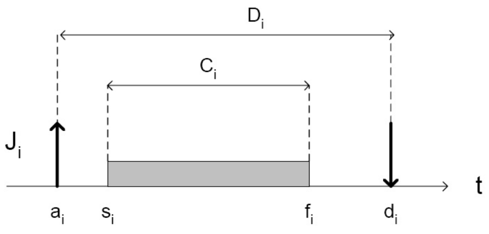


Figure 2.2: Figurative description of real-time job characteristic [31]

There's also the notion of slack time $slack_i = d_i - a_i - C_i$: the maximum time a task can be delayed on its activation to complete within its deadline.

An activation pattern is a rule to generate the sequence of $a_{i\,j}$. Possible examples of activation patterns are:

Periodic the sequence is constructed as follows:

$$\begin{array}{ll} a_{i,j} = \phi_i & \text{if } j = 0 \\ a_{i,j} = a_{i,j-1} + T_i & \text{if } j > 0. \end{array}$$

In this case T_i is said *period* and ϕ_i is said *offset.*

Sporadic the sequence is constructed as follows:

$$\begin{array}{ll} a_{i,j} = \phi_i & \text{if } j = 0 \\ a_{i,j} \geq a_{i,j-1} + T_i & \text{if } j > 0. \end{array}$$

In this case T_i is said *minimum inter-arrival* time and ϕ_i is said *offset.*

In this work we will more generally consider an activation pattern driven by a *timed-automata* that we will specify in the next chapter. This is because in reality there are a lot more possibilities in activation patterns than the few just introduced. An example of a complex activation pattern task is given in figure 2.3. We will enter into the details of the timed automata modelling later on. For the moment being, it suffices to say that this figure describes a task which is first activated after an offset time O_1 and then after period Q_2. Afterwards, non-deterministically, it could either wait for an event or for time Q_1 to be activated, before waiting for the period of Q_2 again.

From this simple example, we can easily see that periodic and sporadic patterns are special cases of timed automata driven activation.

Timing Properties

Real–time tasks can be further classified according to the criticality of their timing constraints. In particular, we can introduce the following taxonomy of timing constraints:

1. Hard deadlines

   An hard deadline requires that all for all the jobs $J_{i,j}$ activated by task τ_i we have $f_{i,j} \leq d_{i,j}$. A violation to any of these constraints, will cause fatal consequences on the system.

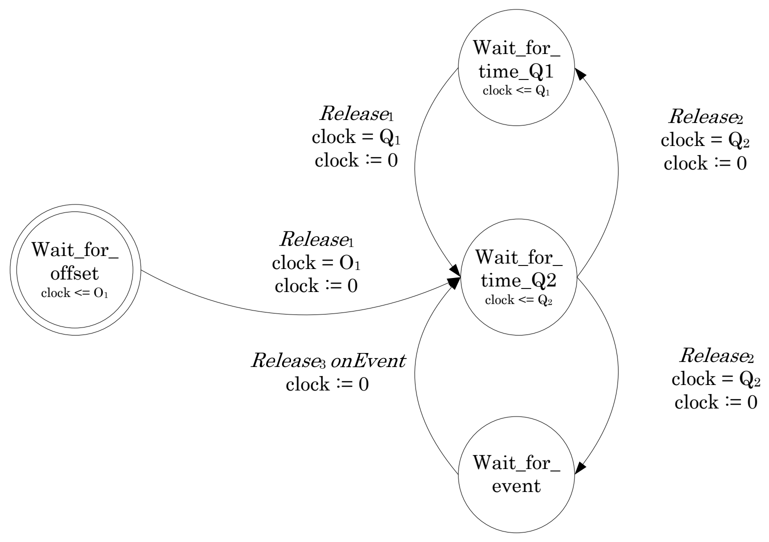

Wait_for_ offset clock <= O1
Release1 clock = O1 clock := 0
Wait_for_ time_Q1 clock <= Q1
Release1 clock = Q1 clock := 0
Release2 clock = Q2 clock := 0
Wait_for_ time_Q2 clock <= Q2
Release3 onEvent clock := 0
Release2 clock = Q2 clock := 0
Wait_for_ event

Figure 2.3: The example of task with complex activation pattern

2. Soft deadlines

   For a soft deadline, some failures can be tolerated up to certain extent. The completion after a deadline decreases the task performance, however it does not entail serious harm to the system. System such as in-flight entertainment system have this characteristic. The overall performance of the audio or video could get poorer on cases where some deadlines are missed, but this will not bring serious effects to the safety of the passengers. The respect of a soft deadline can be measured in probabilistic terms. This opens up the possibility of providing probabilistic real–time guarantees. We can require that the probability of respecting the deadline be grater than a specified value, $Prob(f_{i,j} \leq d_{i,j}) > \mu$, meaning that the percentage of possible deadline failures in the task execution will be bounded by μ. This marks a big difference between a soft real–time task and a best effort task, for which no guarantee whatsoever is offered on the timing performance of the system [16].

3. Firm deadlines

   As in the case of soft deadlines, some failures are tolerated for a firm deadline. However, the required temporal guarantees are expressed in a deterministic fashion. Instead of using probability, the failure measurement can be expressed as the maximum number of deadlines that can be violated consecutively until the system is deemed failed. For example, a system could tolerate up to two consecutive missing deadline events, retaining its functionality in spite of a decreased performance. However, if more than two consecutive deadlines are missed the system will no longer be functioning properly. We categorise this type of constraints as firm deadlines [43].

Interaction between tasks

Two types of interactions are possible between the tasks of a real–time system:

1. Precedence constraints

Some systems require that a job of a task can be activated only after another job of another task terminates. This could happen because a task needs the output of another task as its input. This type of precedence relation is usually described through a directed acyclic graph (DAG).

2. Resource constraints

   Some tasks require, for their execution, access to mutually exclusive shared resources. This type of access is usually supported by appropriate primitives of the Operating System such as mutex region, or semaphores. These shared resources could be a data structure, set of variables, memory area, or set of registers. The access to mutually exclusive resource requires a *blocked* state for the tasks. The presence of a blocked state is important for the real–time performance of the system since it potentially introduces priority inversion [49]: a situation where a real–time task can be delayed for an arbitrarily long time by tasks having a smaller priority.

We define a real–time system as composed of *independent tasks* if all of its tasks have neither precedence constraints nor resource constraints. Most of the application that we will discuss in this thesis cover systems with independent tasks. However, our methodology allows for modelling certain types of precedence constraints between the tasks.

Schedulability of Real–Time System

Because tasks in a real–time systems are allowed to have overlapping execution requests that can exceed the number of available processing units, a *scheduling policy* is required. A scheduling policy is a set of rules whereby the operating system decides, for any point in time, which of the competing tasks should be granted the access to the processing unit. A *scheduling algorithm* is the algorithmic translation of a scheduling policy. A *schedule* is an assignment of tasks to the processor such that each task is executed until completion.

As we are mostly focusing on the feasibility of hard real–time systems, the key property of interest is the following:

Definition 2.1. A task τ_i is said to be *hard real–time schedulable* if for all activation patterns of the tasks in the system and for all the jobs $J_{i,\,j}$, we have $f_{i,\,j} \leq d_{i,\,j}$. The system $\mathcal{S}$ is schedulable if all tasks are hard real–time schedulable.

The problem in every real-time system is how to arrange all the real-time tasks involved in a schedule so that the system is schedulable. In the rest of the thesis we will use the terms *schedulable* and *feasible* interchangeably most of the times since schedulability will be the focal point of our interest. In some limited cases we will consider the combination of schedulability with other property of interests. In this case the term feasibility will take a broader meaning, but this will be clear in the context of the discussion. Further, unless specified otherwise, when we are discussing about schedulability, we refer to the schedulability in hard real–time sense.

In a real–time systems compounded of independent tasks, a task τ_i might be not schedulable for the following reasons:

1. the task execution requirements exceed the power of the CPU, e.g., $\exists j$ such that $a_{i,j} + C_i \geq d_{i,\,j}$.

2. the task undergoes scheduling a high scheduling interference from higher priority tasks (by scheduling interference me mean the interval of time the task is delayed to accommodate the execution of higher priority tasks).

When tasks interact, a task's schedulability could be affected by the blocking time on shared resources.

In a firm real–time system, the schedulability property can be defined as follows :

Definition 2.2. A task τ_i is said to be *firm real–time schedulable* if, for all activation patterns of the tasks in the system, given n adjacent activations $J_{i,j}$ we have at most m times of $f_{i,j} > d_{i,j}$. The system $\mathcal{S}$ is schedulable if all tasks are firm real–time schedulable.

Scheduling algorithm

Central for the attainment of the schedulability properties is the selection of the most appropriate scheduling algorithm. Different families of scheduling algorithms can be identified using the following classification [31]:

1. non preemptive: the scheduling algorithm is not allowed to interrupt the execution of a task: once it is assigned the CPU the task runs to the completion of the job;
2. preemptive : an executing task can be interrupted to yield the processor to another active task that, in that specific moment, has gained a higher priority;
3. static: all parameters (including the scheduling priority) are assigned to tasks before their activation;
4. dynamic: the task parameters can be changed during its execution; therefore a task can have its priority raised or decreased in different moments;
5. offline: the schedule is decided before the task activation;
6. online: scheduling decisions are taken at at run-time

The policy to decide changes in the task scheduling priority is deeply influenced by the scheduling algorithm. For instance, if jobs have a long duration, the choice of a non-preemptive and on-line policy virtually nullifies the assignment of priorities. For online algorithms with static parameters the rate monotonic assignment (i.e.., priority growing with the activation rate) is known to be optimal if the tasks are activated periodically and if deadlines are equal to the period [42].

Task states

The presence of a scheduler requires to associate a task with a state. The possible states are the following:

1. active: a task that can potentially be executed
2. ready : a task waiting for the processor availability
3. running: a task in execution

The possible transitions between these different states are shown in Figure 2.4. First from being inactive a task will be activated when an appropriate event is triggered. After its activation, the task will be in *Ready* state, awaiting to be scheduled by the scheduler. Upon being scheduled, the task will migrate into the *Running* state. At this point, it could either execute until completion or, in case of preemption, it could go back into the *Ready* state. In case the task needs for its execution a shared resource that is locked by some other tasks, it moves to the *Waiting* state until the resource is available. Afterwards, its state will shift to the *Ready* state again until it is scheduled to be executed.

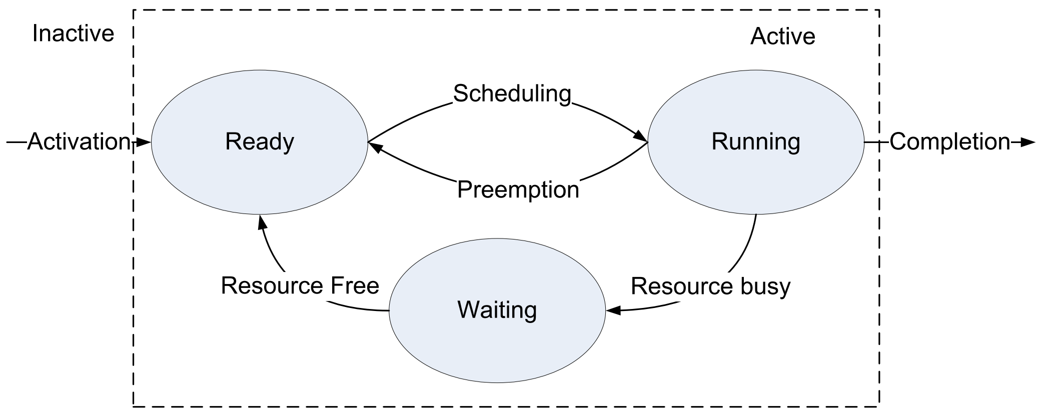


Figure 2.4: The different states of a task

2.2 Schedulability area problem formulation

We are now in condition to formally state the problem addressed in this thesis. First, we need to define design variables and parameters.

Design variables and parameters

As mentioned above, each task in an application can be associated with a set of parameters. Part of them are ground to a fixed value by prior design choice (e.g., the activation time of a control task could be imposed by considerations that have to do with control engineering). The typical real–time scheduling problem are formulated in the case of ground parameters. For instance, considering the case of periodic task sets the typically schedulability problem that can be formulated takes the following form:

Problem 1. *Consider a task set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ scheduled with a given scheduling algorithm and periodic activation pattern. Assume all the tasks have a* given *worst case execution time C_i. Decide whether or not all tasks are schedulable.*

In this scenario, we consider a scheduling algorithm that allows preemption and allocates the CPU to the task using a fixed priority policy.

In the industrial practise, the designer could have a certain freedom of choice for some parameters. We will refer to these as “design parameters”. An example, could be the activation offset of the task, which can sometimes be tweaked to improve the system schedulability. In addition, other “exogenous parameters” have an inherent variability (e.g., due to an insufficient knowledge or to environmental conditions).

In the example reported in Problem 1, the computation time could be considered as a design parameter in many realistic scenarios. In principle, a designer can change this parameter in two ways: 1) changing the implementation of a task, 2) changing the power of the hardware (which, roughly speaking, operates as a scaling factor). In a different context, the computation time could be regarded as an “exogenous parameter”; for instance, the computation time could be related to a legacy application which is known to a limited extent. Other design parameters could be hidden underneath the activation pattern.

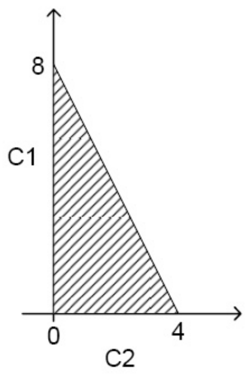


Figure 2.5: The example of the feasibility region $\mathcal{R}$

For instance we could have variable offsets for a periodic task execution. Generally speaking we associate to each task τ_i a vector $\boldsymbol{\rho}_i$ of design parameters (e.g., for a periodic task with offset this vector is given by: $\rho_i = [\phi_i, C_i]$).

If the role of parameters is considered a problem like Problem 1 takes a different and more general form.

2.2.1 General Problem Formalization

First of all, it is possible to define the feasibility region as follows:

Definition 2.3. Consider a task set $\mathcal{S} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, scheduled by a given scheduling algorithm. Let each task τ_i be associated to a vector of design parameters ρ_i taking values in the set Γ_i and let $\boldsymbol{\rho} = [\rho_1, \rho_2, \ldots, \rho_n]$ be a vector aggregating the parameters of all the different tasks. The feasibility region $\mathcal{R} \subseteq \Gamma_1 \times \Gamma_2 \times \ldots \Gamma_n$ is defined as: $\boldsymbol{\rho} \in \mathcal{R}$ if and only if the system is schedulable with the choice of parameters $\boldsymbol{\rho}$ for the different tasks.

The definition above leads us to the following formalisation.

Problem 2. *Consider a task set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ scheduled with a given scheduling algorithm and with activation pattern of each τ_i specified. Let ρ be the vector of design parameters defined in the set $\Gamma = \Gamma_1 \times \ldots \times \Gamma_n$. Find the feasibility region $\mathcal{R}$.*

The definition and the problem formulation above are easily generalised to systems using multiple types of resources (processors, communication links) and to the verification of more general temporal properties than the simple task schedulability.

Example 2.1. *Consider the example of a task set $\mathcal{S} = \{\tau_1, \tau_2\}$ where τ_1, τ_2 are two periodic tasks with $T_1 = 4$ and $T_2 = 8$. Their deadlines are equal with their periods and their offsets are zero, i.e., $D_1 = T_1 = 4$, $D_2 = T_2 = 8$, $\phi_1 = \phi_2 = 0$, and the design parameters are $\rho_1 = [C_1]$, $\rho_2 = [C_2]$. In this case the choice of parameters $\boldsymbol{\rho} = [\rho_1, \rho_2]$ is described in the equation $\frac{C_1}{4} + \frac{C_2}{8} \leq 1$ [31]. Therefore the region $\mathcal{R}$ can be calculated and can be seen in figure 2.5.*

2.3 Example Scenarios

To illustrate the potential applications of the methodology in a design flow for embedded systems, we propose a couple of simple possible real world design scenarios. Later on in the thesis, some of this scenarios will be reprised and used as a running examples for our discussion. In this context, we simply use them to outline the possible design issues that our methodology allows us to address and the type of results that can be obtained.

Scenario 1

Given of a real-time system with a set of periodic tasks and offset to each tasks arrival rate such as a control system in nuclear power plant. The timing deadlines are hard deadlines as this is a critical system. The tasks are scheduled using a pre-emptive scheduler with static pre-assigned priorities. The designer is required to maximise the system robustness, allowing the system to survive in case of unforeseen environmental disturbance that could affect its computation time.

We use a periodic task system $\mathcal{S}_1 = \{\tau_1, \tau_2\}$. Periods are fixed and chosen equal to the relative deadline: $\mathbf{T_1} = \mathbf{D_1} = 20$ and $\mathbf{T_2} = \mathbf{D_2} = 30$.

Use case 1: Consider a design scenario in which the computation times are not known with a sufficient precision and the system is very close to the border of the schedulability region found with traditional real-time scheduling theory (i.e., assuming null offset). The design problem is how to choose a positive offset O_2 so as to maximize the schedulability region (and hence the robustness of the system). Thereby, we aim to compute the schedulability region with respect to the following set of free parameters: $\{C_1, C_2, O_2\}$ fixing $O_1 = 0$.

The application of the methodologies shown below in the thesis allows us to construct the region illustrated in Figure 2.6. We can see three different sections obtained by cutting the three-dimensional region with the planes $O_2 = 0$, $O_2 = 5$, and $O_2 = 8$. In accordance with the notion of critical instant from the classical real–time scheduling theory, $O_2 = 0$ corresponds to the smallest schedulability region. The choice $O_2 = 8$ is the one that ensures the maximum robustness for the choice of C_1, while O_2 corresponds to the greatest schedulability region.

Use case 2: Assume that the worst computation time of the tasks are fixed, $\mathbf{C_1} = 11$ and $\mathbf{C_2} = 12$. This task system is not schedulable with zero offsets as it is possible to see with a standard response time analysis as can be seen in the upper illustration in Figure 2.7 Therefore, we would like to identify the possible values for the offsets O_1 and O_2 that make the system schedulable.

The region resulting from the application of our methodology is shown in Figure 2.8. Using the information from the figure we can then simply pick the values for O_1, O_2 and O_3 inside the feasibility region to make $\mathcal{S}_2$ schedulable. We can choose $O_1 = 5$ and $O_2 = 1$ for example. And the system will now be schedulable, in spite of a system utility close to 95% and to a non–harmonic choice of the periods.

Scenario 2

This example is inspired by the Heterogeneous Communication System (HCS) of a civil aircraft [27]. The problem is to stream the audio package and synchronize them. The architecture of the system is shown in Figure 2.9.

The HCS system contains a common server, wired and wireless communication networks and a number of devices. The components of the network communicate through Network Access Controllers (NAC), which perform gateway function and routing, and are connected in a daisy chain topology. We focus on audio devices which are required to distribute music and audio announcements to the main cabin. The audio stream is transmitted by the server through the network. To avoid echo effects, the devices must reproduce the audio at synchronized instants. For this reason, the network, server and

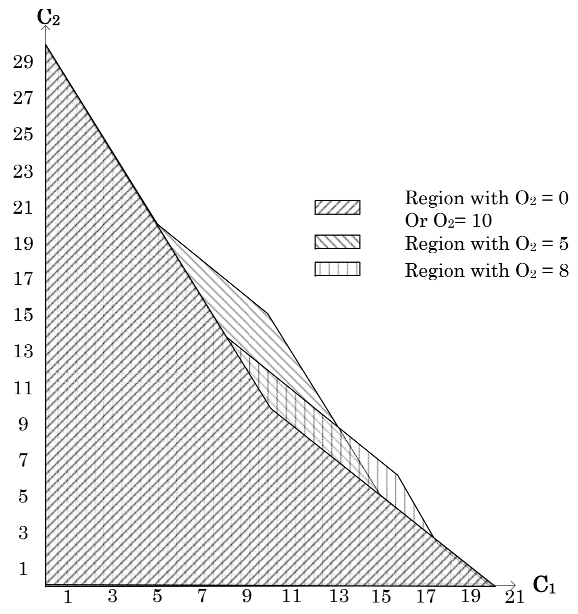


Figure 2.6: Feasibility regions of Scenario 1, Use case 1,in the domain of C_1 and C_2 for different values of O_2

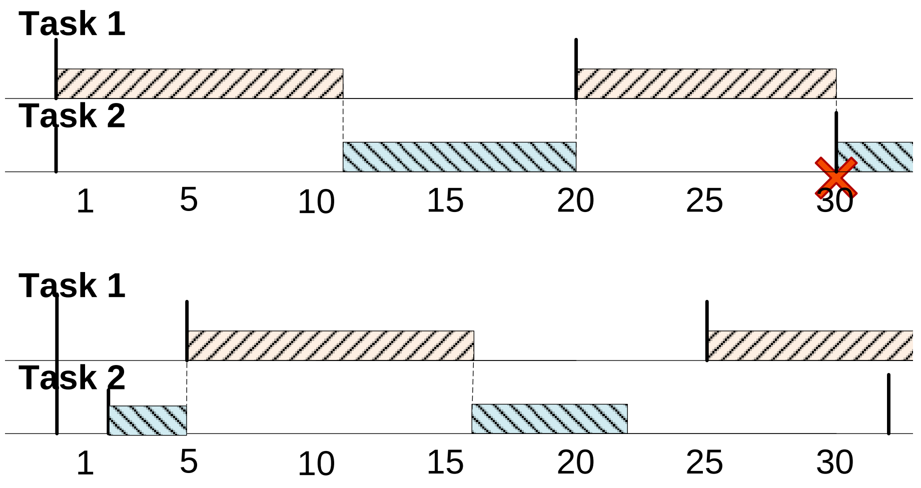


Figure 2.7: In the upper figure the system will fail using $O_1 = O_2 = 0$, while in lower figure the system will succeed using $O_1 = 5$ and $O_2 = 1$

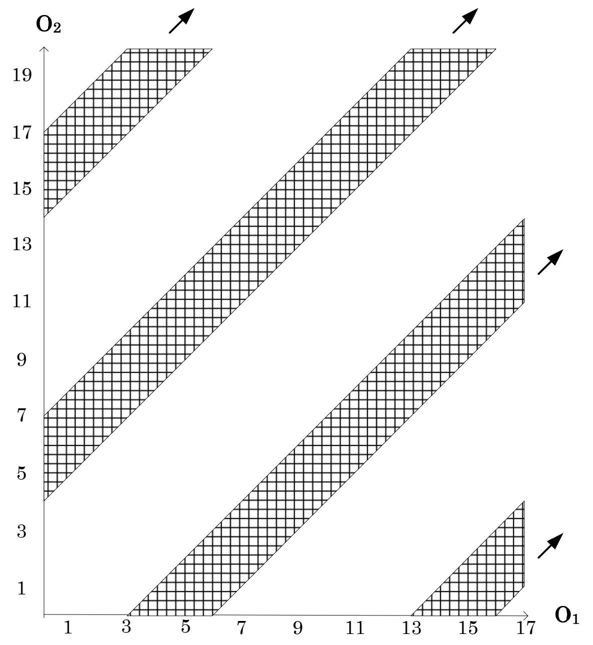


Figure 2.8: Regions of feasibility in the domain of O1 and O2 for Scenario 1, Use case 2.

devices implement also the Precision Time Protocol (PTP) [47] to synchronize their clocks.

The communication between the server and device is asynchronous. The server sends an audio packet every $audPeriod$ ms; audio packets are characterized by two parameters: a sequence number i and a timestamp ti denoting the time the packet has to be played at the device. Due to varying network conditions, packets arrive at the device (except if they are lost) with a minimal latency $Lmin$ and a maximal latency $Lmax$. The NACs simply forward the incoming packets to the devices. Packets passing through the NACs experience delay of $Lnac$ during which they are preprocessed by the NACs. The device processing time for each audio packet is τ, after which the device is ready to receive the next audio packet. The PTP protocol runs on the server, the devices and NACs, and is used to synchronize the respective clocks. The audio packets have firm deadline requirements. A packet may miss its deadline as long as the previous packet has not already missed the deadline.

Various timing delays are to be guaranteed, for example, in a scenario where two devices are connected to the server, it should be guaranteed that both devices are synchronized within an error of 0.1 ms (synchronization precision).

In this system the parameters that we can play with for example are the buffers used, the latency $Lmin$ and $Lmax$ in the medium and also $Lnac$ in NAC, the computation time of each processes, and the clock drift c which is the offset time of the local clock compared to the server clock. The properties that we can check are for example:

- Whether or not buffer overrun could happen.
- That the firm deadlines of the audio packets are not missed
- The maximum possible latency in the medium that still guarantees the system will not fail.

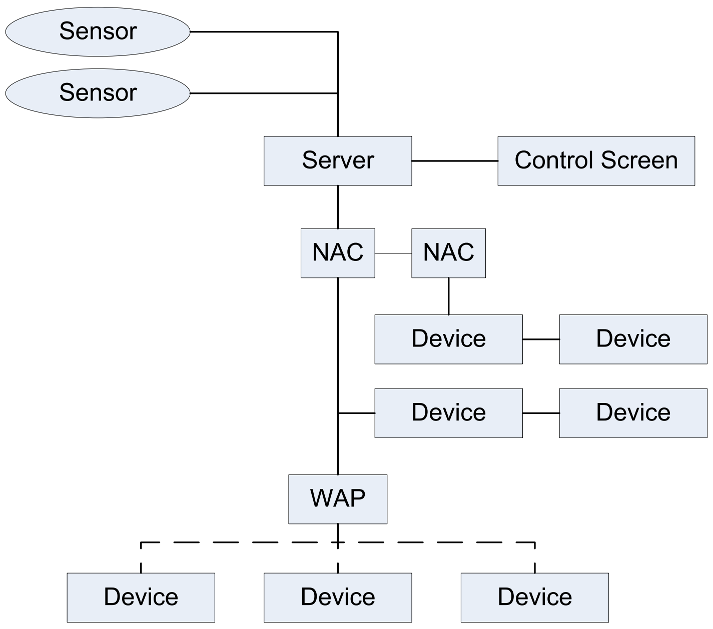


Figure 2.9: Heterogeneous Communication System (HCS)

- Can we tweak the computation time of task to enable maximum robustness of system schedulability?

The objective of this work is to answer the last query for example by something like is shown in Figure 2.10, where we can see the possible values for the computation time that will ensure the schedulability on different values of clock drift Δ.

Scenario 3

Consider a real–time system with CPUs whose speed changes with the size of the backlog buffer. The scheduler of the example load-dependent frequency adaptation scheduler is adapted from [35]. The scheduler can change its processing speed when the number of tasks in the buffer has exceeded certain threshold. The tasks are activated periodically with jitter. However, unlike in the previous scenarios, the task activation and the result in this case is represented through timed automata that produces *stream*.

A stream is characterized by a pair of so called arrival curves [52]. These arrival curves bound the number of activation events for any interval $\Delta \in [0, \infty)$. Thus arrival curve is characterized by the maximum number of tasks in burst and the widths Δ of the stairs. We need to capture the system behaviour given the input and output of the system in terms of the arrival curve.

The parameters that we could play with in this kind of systems are the maximum number of tasks in burst on the input and or the output of the system, the widths of input/output curve stairs, maximum latency that can be tolerated for each task activated, and its maximum buffer size.

The type of query that we could have is for example what is the schedulability region for the BMAX-One, the maximum burst number of activated tasks, and DeltaOne, the widths of input curve stairs? The result that we expected for this query is illustrated in Figure 2.11.

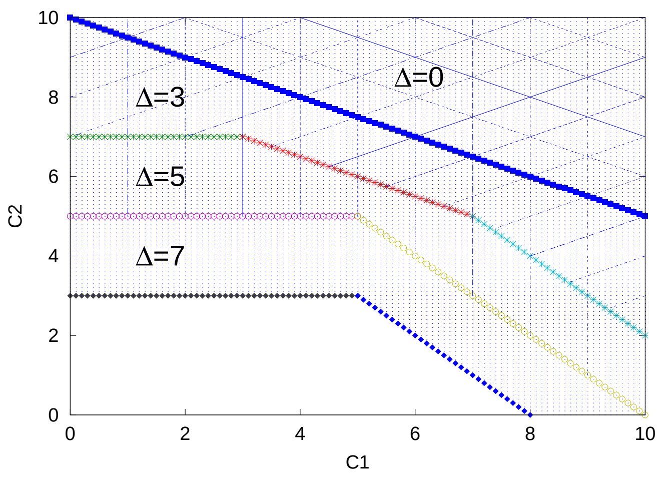


Figure 2.10: Audio feasibility and error regions for $\Delta = 0, 3, 5, 7$.

2.4 Challenges

Concluding from the illustrations in the scenarios presented previously, here are the challenges that we need to handle in designing real–time sytem.

1. Providing model that could represent general activation pattern of tasks and scheduling requirements and policies.
2. Performing the schedulability analysis for all possible design parameters

Our main challenge is to have a tool that is able to answer both challenges we presented above.

In our research we need to handle these outstanding issue on the model and the method that we will use to analyse the feasibility of a system:

- Model

  The model that we use need to fulfil the following criteria:

  1. Expression power
     The model should have strong expressive power to be able represent general real time system.
  2. Usability
     The model should be easy enough to use by designers that have no in-depth knowledge of formal methods.
  3. Tractability
     The model should be tractable as in having other use other than just plain for modelling. For example it could also be used as problem presentation.

- Verification Method

  The verification method that we use needs to be general. It should be able to analyse all possible parameters that exist in the model. The method should be able to output meaningful information to the user regarding the possible parameter valuations of the system and the subsequent schedulability of the system.

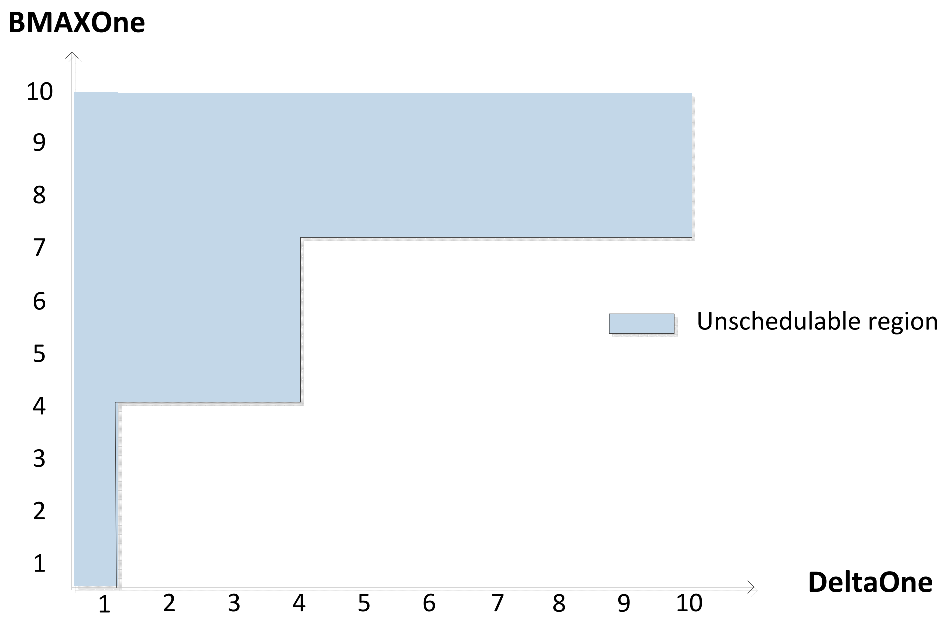


Figure 2.11: Schedulability region of system

Chapter 3

Background Knowledge

3.1 Satisfiability Modulo Theory

3.1.1 Logic Terms

First, we enumerate several definitions that we will use in discussing the topic of Satisfiability Modulo Theories (SMT) and logic in general.

Propositional Logic

We start from the most basic building block of logic formulas: propositional variable. A *propositional variable* is a variable with value of either **True** or **False**. A *propositional formula* φ is built from propositional variables. It can consist of one propositional variable p, or a negation $\neg\varphi_0$, conjunction $\varphi_0 \wedge \varphi_1$, disjunction $\varphi_0 \vee \varphi_1$, implication $\varphi_0 \rightarrow \varphi_1$ or bi-implication $\varphi_0 \leftrightarrow \varphi_1$ of other propositional formula φ_0 and φ_1. A literal is a propositional variable p or its negation. Literals are connected by disjunction to form a clause. We say that a formula is in conjunctive normal form (CNF) if it consists of clauses connected by conjunction $C_1 \wedge \ldots \wedge C_n$. Any formula can be made into a CNF formula.

A *truth assignment* M for a formula φ assigns all the propositional variables in φ to either **True** or **False**. A truth assignment M *satisfies* φ, with the notation $M \vDash \varphi$, if the assignments of values in M makes φ **True**. A formula φ is satisfiable if there is an M that could satisfy it, $M \vDash \varphi$.

EXAMPLE Given a formula $\varphi = (p \wedge \neg q) \vee r$, the truth assignment
$M = \{p \mapsto True, q \mapsto False, r \mapsto False\}$ satisfies φ.

Satisfiability Problem

Given a propositional formula, the satisfiability problem amounts to deciding if there is a model M that satisfies it. We call this specific problem as *propositional satisfiability SAT problem* or shortly as *SAT problem.* The solution of a SAT problem is NP-complete.

First-Order Logic

First-order logic extends the propositional logic with functions, predicates and quantification. The basic entities in first-order logic are terms. A *term* t can be a constant c, a variable x over a certain domain, or a function whose arguments are terms. The notation for an n-ary function f is $f(t_1, \ldots, t_n)$. Predicates can be seen as functions that return either **True** or **False**. A predicate describes the relation between its arguments. For example $P(x, y)$ could mean x smaller than y and $Q(y)$ could mean y is a female.

A *formula* is a predicate $p(t_1, \dots, t_n)$ or negation $\neg\varphi_0$, conjunction $\varphi_0 \wedge \varphi_1$, disjunction $\varphi_0 \vee \varphi_1$, implication $\varphi_0 \Rightarrow \varphi_1$, bi-implication $\varphi_0 \Leftrightarrow \varphi_1$, and quantification $\forall x\varphi_0$ or $\exists x\varphi_0$ of other formula φ_0 and φ_1. A Σ-formula φ is a formula with all the symbols defined in the domain Σ. A *free variable* x in φ is a variable that is not bounded by any quantifier $\forall$ or $\exists$. A *sentence* is a formula without free variables.

A *model* M is a set of interpretation of all the variables, functions, and predicate symbol that makes the formula φ **True**. A formula is satisfiable is there is a model M that could satisfy it, $M \vDash \varphi$.

Theory

A Σ-*theory* is a set of sentences over Σ. The satisfiability of a theory T is *decidable* if there is a procedure that determines the satisfiability of any quantifier-free formula in it. Linear Arithmetic theory for example is the theory with functions $+$ and $-$ applied to either numerical constants or variables, and multiplication between a constant and a variable. The predicates of this theory are $=, <, \leq$. Given a linear arithmetic formula, its satisfiability can be decided by using dual simplex algorithm for formula with = and $\leq$ predicate and by using the algorithm extension for formula with strict inequality $<$ predicate. Since there exists this algorithm to determine the satisfiability of formula in Linear Arithmetic theory, this theory is decidable.

Example 3.1. *We provide an example formula of first-order logic over linear arithmetic. The formula* $\varphi = (x + 2 * y \leq 0) \wedge (x < 0 \vee (2 * x + 4 * y + 2 < 0))$ *is made of Boolean combinations of linear arithmetic constraints. The model* $M = \{x = -1.0, y = 0.5\}$ *satisfies* φ*. Thus, formula* φ *is satisfiable.*

3.1.2 Satisfiability Modulo Theory

Many of the industrial problems require the expressive power of first order logic. Aside from simple declarative propositions, predicates and quantification are usually needed. And, instead of having all boolean variables an industrially relevant problem could have a mix some real and or integer variables and some arithmetic operations. For instance, in our problem specification in modelling real– time system, we need to include the notion of continuous time in the environment as the rationals factor in the formula.

Linear programming algorithms can check the satisfiability of conjunctions of linear arithmetic inequalities, but they can not be applied directly to Boolean combinations. For this kind of problems we define the notion of Satisfiability Modulo Theory (SMT). A formula φ is *satisfiable modulo T* if $T \cup \varphi$ is satisfiable. Referring to the problem in Example 3.1 the formula is satisfiable modulo linear arithmetic theory if the formula made of the union of solving the arithmetical constraints in the theory of linear arithmetic and the boolean parts of φ is satisfiable.

Generally, SMT is a decision problem involving various theories which are expressed as simple first-order logic. SMT solvers work by combining special purpose algorithms for each domain. Example of these theories are, e.g., the theory of difference logic $\mathcal{DL}$, the theory $\mathcal{EUF}$ of equality and uninterpreted functions, and the quantifier-free fragment of Linear Arithmetic over the rationals $\mathcal{LA}(\mathcal{Q})$ and over the integers $\mathcal{LA}(\mathcal{Z})$ [15]. This enables a richer representation than simple boolean SAT formulas.

First-order logic is however undecidable. Therefore, most procedures for SMT focus on efficiently solving certain scope of problems that occur in practices. SMT solvers rely on the alternating work of

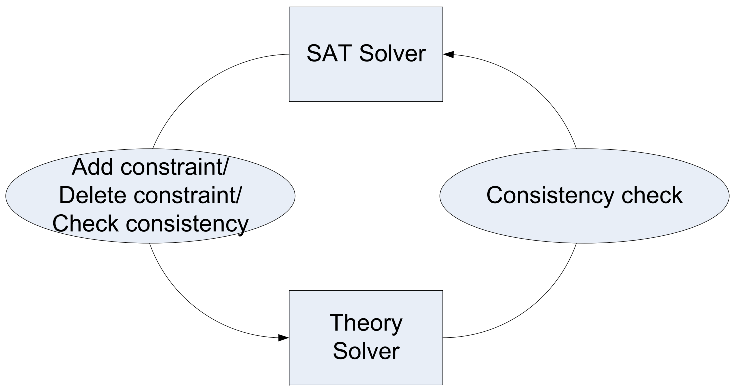


Figure 3.1: A simple description on SMT solving schema

a SAT solver to solve the Boolean parts of the formula, and of Theory solver(s) to solve the other logic problems involved. The interaction between the two class of solvers is illustrated in Figure 3.1. [1].

An SMT-solver first will extract the Boolean skeleton on the input formula. This is obtained by replacing the theory constraints in the formula with auxiliary Boolean variables. Given the Boolean skeleton formula of φ, SAT solver performs *systematic search* in a tree of which each vertex represents a propositional variables and the outgoing edges represent **True** and **False**. This systematic search is performed to find a truth assignment M that will satisfy the Boolean skeleton of φ. Most search based SAT solvers are currently using DPLL approach. While the SAT-solver tries to construct a satisfying assignment for the Boolean skeleton of the SMT formula iteratively, the theory solver then verifies the conjunction of the constraints which have to hold according to the so far constructed assignment.

Example 3.2. *We refer to the example problem given in example 3.1. If we replace each of the arithmetical constraints in $\varphi = (x + 2 * y \leq 0) \wedge (x < 0 \vee (2 * x + 4 * y + 2 < 0))$ by a Boolean variable we will obtain the formula $A \wedge (B \vee C)$ as the Boolean skeleton of φ. First the SAT Solver will try to assign $A \mapsto True$. This will send the constraint $x + 2 * y \leq 0$ to the theory solver. Theory solver will check its consistency to the* ***True*** *assignment and will report that it is satisfiable. SAT solver then continues to assign $A \mapsto True, B \mapsto True, C \mapsto True$. The rest of the formula φ is now sent to the theory solver. The theory solver will report unsatisfiability and give back the constraint A, $x + 2 * y \leq 0$, and C, $2 * x + 4 * y + 2 < 0$, as reasons. The SAT solver uses this information to undo a part of the current assignment (backtrack), learn that it should not assign both A and C to True, and continue the search.*

3.1.3 Bounded Model Checking

The approach we used in this work to carry out the verification process of SMT is symbolic and is based on bounded model checking (BMC) method [9]. The verification of the timed system is done by considering a finite and increasing time bound. In each iteration, the verification problem is transformed into the satisfiability problem of a set of formulas defined over boolean variables and real variables. We will provide an example of how a timed automata is unrolled step by step for BMC on the timed automata section example 3.4.

[1]Adapted from http://www-i2.informatik.rwth-aachen.de/i2/simplex_in_smt/

The result of the BMC is a satisfiability response or a counterexample. Namely, for a finite transition system *M* the property of interest is the satisfiability of a Linear Time Logic (LTL) formula φ. For a given number of steps *k*, the model checker searches for counterexamples of length *k* as the bound, and decide that *M* satisfies φ iff there is no counterexample exists in the path of length *k* or less than k.

The most important problem of BMC is to identify a number of steps CT such that if no counterexamples are found within the bound *CT* then *M* $\models \varphi$ (the formula is satisfied). Such problem is defined as the *Completeness Threshold (CT)* problem. In [24] a computation method to find an upper bound of *CT* is described for the reachability problem on finite systems. The bound is sufficiently computed by finding the initialized diameter $d^I(M)$, which is the longest shortest path from an initial state to any reachable state of *M*. Using Büchi Automata with Vardi-Wolper LTL model checking framework it is found that such value of *CT* is computable with a formula of size $O(k \log k)$. An important open issue is to verify whether a similar notion on the computability of *CT* can also be applied to timed automata.

A problem with the theoretical bound just mentioned is that it tends to be un-necessarily large. In practical cases, the maximum length of a possible counter-example for the property of interest can be much shorter. Very useful in this respect is to complement BMC methods with inductive reasoning. Inductive reasoning is a reasoning method which work on verifying that the initial state satisfies property φ, and from all the steps from 0 to k that verifies the property, we can only go to step $k + 1$ that also verifies the property. Optimizing BMC with inductive reasoning we could verify if no counter example is found at step k then we will not found it on $k + 1$ step, thus providing termination result to the BMC search.

3.1.4 Tools

One of the tool developed to decide the satisfiability of theories problem belonging to the category of SMT is MathSAT [13]. MathSAT solves this problem based on Davis-Putnam-Logemann-Loveland algorithm (DPLL), which decides the satisfiability of propositional logic formulae in conjunctive normal form(CNF) by using backtracking decision procedure, and it is able to solve SMT problem for various theories. One of them, which is important for our purposes, is Linear Arithmetic over the Reals (LA (R)). The ability to solve problems based on this LA (R) theory encouraged the use of MathSAT in this work.

This MathSAT is a foundational tool for NuSMV3, the latest version of NuSMV [22], a verification tool that combines the strengths of Binary Decision Diagrams (BDDs) of NuSMV and SMT solvers of MathSAT. NuSMV itself is a reimplementation and extension of SMV symbolic model checker, the first model checking tool based on Binary Decision Diagrams (BDDs). BDD is a data structure that is used to represent a Boolean function. Abstractly, BDDs are as a compressed representation of sets of boolean functions. However, unlike other compressed representations, operations are performed directly on this compressed representation, thus minimizing the complexity.

The extension of NuSMV to NuSMV3 entailed the definition of a language involving new types involving both Integer and Reals and hybrid automata. The full integration with MathSAT SMT solver has also enabled the construction of other model checking algorithms which were not in the initial distribution of NuSMV. Most notably the model checking that we will use are Bounded Model Checking via SMT and Interpolation based invariant checking for induction. An extended set of features such as requirements analysis, dependability assesment, and safety analysis are also provided in this extension.

A comprehensive description of the tools exceeds the purposes of this thesis. The interested reader is referred to the documentation present in the tool website (https://es.fbk.eu/tools/nusmv3/). In the rest of the chapter, we will simply describe how the tool will be applied for our purposes: the modelling and

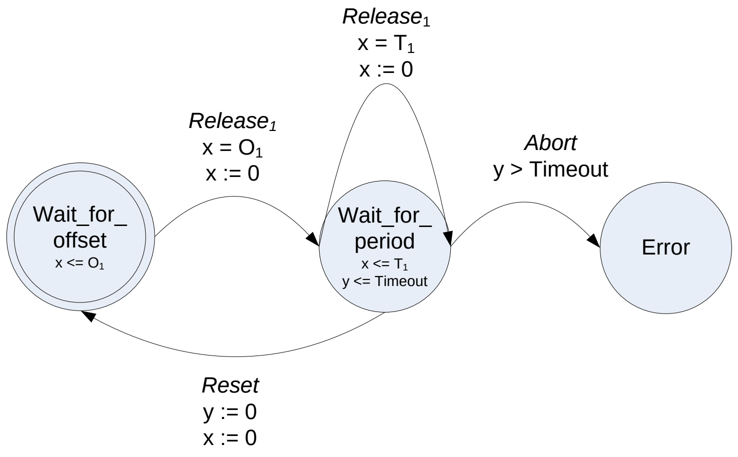


Figure 3.2: TA_1, an example of a task activation automata to be encoded in NuSMV.

analysis of timed automata.

3.2 Timed Automata

The notion of timed automata was first proposed by Alur et al. [1] to enable an explicit modelling of time, and it was later recognized as an effective formal notation to model the behaviour of real-time systems. A timed automata is a directed graph, in which nodes are called the locations and the arcs are called the transitions. Time modelling is achieved by the use of clock variables. The constraints on the values of the clock determine the temporal evolution on the timed automata.

3.2.1 Timed Automata Formal Notation

A timed automaton A [1] is defined as a tuple of $\langle L, L_0, \Sigma, X, I, E\rangle$ and its components are defined as follow:

- L defines the finite set of locations in the timed automata.
- $L_0 \subseteq L$ defines the set of initial locations
- Σ defines a finite set of labels of events affecting the evolution of the timed automata.
- X defines a finite set of clocks
- I, describes the invariant of a location. It defines a mapping between each location $s \in L$ with some clock constraint in the set of clock constraints $\Phi(X)$
- $E \subseteq L \times \Sigma \times 2^{\mathcal{X}} \times \Phi(X) \times L$ defines the set of switches. A switch with the formal representation $\langle s, a, \varphi, \lambda, s'\rangle$ describes a transition from location s to s', with $s, s' \in L$ on an event $a \in \Sigma$, with clock constraint φ on X as the guard of the switch, and with a set of clock reset action taken upon its switch as defined in λ.

Example 3.3. *We present in Figure 3.2 an example of timed automata TA_1. The components of TA_1 are:*

1. *L_i : $L = \{Wait\_for\_offset, Wait\_for\_period, Error\}$.*
2. *L_0 : $L_0 = Wait\_for\_offset$.*
3. *Σ : The labels are $\Sigma = \{Release_1, Reset, Abort\}$.*
4. *X : The clocks in this example are $X = \{x, y\}$.*
5. *I : We have the invariant for locations $Wait\_for\_offset$ and $Wait\_for\_period$. We write them as $I = \{state = Wait\_for\_offset \rightarrow (x \leq O_1)\} \wedge$ $\{state = Wait\_for\_period \rightarrow (x \leq T_1 \wedge y <= Timeout)\}$.*
6. *E : There are 4 transitions in our example, as we need to name each transition, we will name them with the following format*

   ```
   <origin_state>_<destination_state>_<transition_number>}
   ```

   The set of transitions that we have is

   $$E = \{wait\_for\_offset\_wait\_for\_period\_1,$$
   $$wait\_for\_period\_wait\_for\_period\_2, wait\_for\_period\_wait\_for\_offset\_3,$$
   $$wait\_for\_period\_error\_4, .$$

A system made of a set of automaton is defined as a product construction of the involved automata. Given two timed automata $A_1 = \langle L_1, L_{10}, \Sigma_1, X_1, I_1, E_1\rangle$ and $A_2 = \langle L_2, L_{20}, \Sigma_2, X_2, I_2, E_2\rangle$. Assuming clock sets X_1 and X_2 are disjoint, a product A of A_1 and A_2 is the timed automaton $A = \langle L_1 \times Ł_2, L_{10} \times L_{20}, \Sigma_1 \cup \Sigma_2, X_1 \cup X_2, I, E\rangle$ with $I(s_1, s_2) = I(s_1) \wedge I(s_2)$. The switches are :

1. $\langle(s_1, s_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (s_1', s_2')\rangle$, for $a \in \Sigma_1 \cap \Sigma_2$ and for every $\langle s_1, a, \varphi_1, \lambda_1, s_1'\rangle$ in E_1 and $\langle s_2, a, \varphi_2, \lambda_2, s_2'\rangle$ in E_2.
2. $\langle(s_1, t), a, \varphi_1, \lambda_1, (s_1', t)\rangle$, for $a \in \Sigma_1 \backslash \Sigma_2$ and for every $\langle s_1, a, \varphi_1, \lambda_1, s_1'\rangle$ in E_1 and every t in L_2
3. $\langle(t, s_2), a, \varphi_2, \lambda_2, (t, s_2')\rangle$, for $a \in \Sigma_2 \backslash \Sigma_1$ and for every $\langle s_2, a, \varphi_2, \lambda_2, s_2'\rangle$ in E_2 and every t in L_1

Thus, a transition of A is obtained by coupling the transitions of the individual automata labelled with consistent event sets. A special type of transition that preserves clocks and locations is defined as *stutter transition*. An automata takes the stutter transition when a transition takes place in another automata that is not synchronized with any of its label.

A run of timed automata consists of states and transitions between them. A state is (s, ν), with $s \in L$ and ν assignment to the clocks such that $\nu \models I(l)$. A state is an initial iff $s \in L_0$ and $\nu(x) = 0$ for all clock $x \in X$.

The timed automata transitions can be differentiated into two classes of transitions :

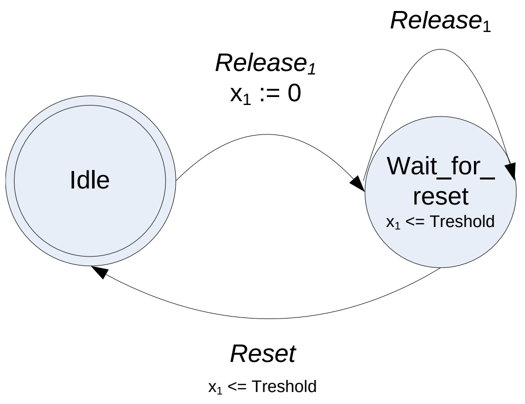


Figure 3.3: A timed automata to be encoded in composition with TA_1

1. Discrete transitions

   It is a transition that happens when in the state (s, ν) a switch $\langle s, a, \varphi, \lambda, s' \rangle$ such that $\nu \models \varphi$ is taken. We write this $(s, \nu) \xrightarrow{a} (s', \nu [\lambda := 0])$. Stutter transition is also categorized in this class.

2. Continuous transitions

   Defined as δ-transitions, they represent the time elapse in the timed automata system. For a state (s, ν) experiencing time elapse $\delta_t \geq 0$, we write $(s, \nu) \xrightarrow{\delta_t} (s', \nu + \delta_t)$ if for all $0 \leq \delta_t{}' \leq \delta_t$, $\nu + \delta_t{}' \models I(s)$. The time elapse transitions are synchronized on all timed automata in a system.

A run of length k is a sequence of states $(l^0, \nu^0), (l^1, \nu^1), \ldots, (l^k, \nu^k)$ iff the first state is initial, and for $i = 0, \ldots, k-1$ it holds that $(l^i, \nu^i) \xrightarrow{\sigma} (l^{i+1}, \nu^{i+1})$ for some switch σ, or $l^i = l^{i+1}$ and $(l^i, \nu^i) \xrightarrow{\delta_t} (l^{i+1}, \nu^{i+1})$.

Example 3.4. *To describe the composition of automata and the run of automata, we present the second example TA_2 in Figure 3.3 which will be synchronized with TA_1, our previous example in figure 3.2. This timed automata TA_2 synchronizes with TA_1 on the event of $Release_1$. After the first event of $Release_1$, within time $Threshold$, TA_2 could non-deterministically fire the $Reset$ event.*

Using the following assignment $\{T_1 = 4, O_1 = 2, Timeout = 9, Threshold = 5\}$, then one of the possible run for the automata TA which is the product of timed automaton TA_1 in Figure 3.2 and timed automaton TA_2 in Figure 3.3 is:

$$\begin{array}{llll} ((\texttt{WFO}, \texttt{Idle}), \{x = 0, y = 0, x_1 = 0\}) & \xrightarrow{\delta_2} & ((\texttt{WFO}, \texttt{Idle}), \{x = 2, y = 2, x_1 = 2\}) & \xrightarrow{\texttt{Release}_1} \\ ((\texttt{WFP}, \texttt{WFR}), \{x = 0, y = 2, x_1 = 0\}) & \xrightarrow{\delta_4} & ((\texttt{WFP}, \texttt{WFR}), \{x = 4, y = 6, x_1 = 4\}) & \xrightarrow{\texttt{Release}_1} \\ ((\texttt{WFP}, \texttt{WFR}), \{x = 0, y = 6, x_1 = 4\}) & & & \end{array}$$

with WFO = $Wait\_For\_Offset$ state and WFP = $Wait\_For\_Period$ in TA_1 while WFR = $Wait\_For\_Reset$ in TA_2. The run above depicts how it starts in the initial location (WFO,IDLE) *which is made of the initial location of TA_1 and TA_2. All the clocks, $\{x, y, x_1\}$, which are the union of clocks from TA_1 and TA_2, have zero values in this initial state. The first transition is a δ-transition of 2 times unit which takes the system to the second state where it has the same location* (WFO,IDLE) *but all the clocks are incremented by 2 time units. The synchronized switch with label $Release_1$ composed of*

$wait\_for\_offset\_wait\_for\_period\_1$ in TA_1 and $idle\_wait\_for\_reset\_1$ in TA_2 is enabled and taken. This discrete transitions changes the location of the automata to (WFP,WFR) *and reset the clocks x in TA_1 and x_1 in TA_2 while the clock y in TA_1 retains its value. Next, we have another δ-transition with 4 time units that increment all the clocks value. And now as the guard of transition with label $Release_1$ that is composed of $wait\_for\_period\_wait\_for\_period\_1$ for TA_1 and stutter transition in TA_2 is enabled. Taking this transition moves us to the next state where we stay in the same location* (WFP,WFR) *but the clock x of TA_1 has been reset while clock y and x_1 retain their values.*

3.2.2 Timed Automata Verification in NuSMV

A first possibility for verifying timed systems is the application of such model checkers as UPPAAL [2]. This approach is taken in the TIMES tool [28, 29, 45], which is a derivation of UPPAAL specifically targeted to the verification of real-time scheduling properties. The approach can be classified as an explicit state model checking and it uses the mathematical properties of Difference Bound Matrices (DBM).

We do not proceed with further details on this since we will take a different approach based on symbolic model checking using NuSMV. This approach performs the analysis of system on the encoding of the system by a set of symbolic formulas and enables the representation of a large state space with moderate memory requirement.

The following aspects are captured in NuSMV symbolic encoding:

1. The structure of the automata
2. The evolution of the automata
3. The composition of automata
4. The required properties

The structure of the automata

A timed automaton A associated with the tuple $\langle L, L_0, \Sigma, X, I, E\rangle$ is modelled in NuSMV as a module. The definition of the automata structure is as follows:

1. Timed automata declaration

   We declare the timed automata by declaring a MODULE with the name of timed automata and input variables of the timed automata. The input variables are variables that will be used globally. One of them is the global variable $delta_t$ which represents the amount of time units that elapsed at each transition. This variable will be declared in the main module and then as input variable to each of the automata.

2. Location declaration

   L is represented by a local variable *state* which takes value in the enumeration of all possible states $s_i \in L$. L_0 is represented by the INIT statement, assigning the variable *state* to one of the location in L that determines the initial location.

[2](http://www.uppaal.com/)

3. Clock declaration

   For each clock $X_i \in X$ a variable X_i with real type is declared. The real type in NuSMV represents the rational values. Operation on time is specified through assignment commands which will be executed each time a transition is taken. This assignment command involves the clock reset when some transition is taken and clock value addition of $delta\_t$ on the transition of time elapse. We will discuss the operation on each clocks on the timed automata evolution encoding.

   We need to specify the initial condition for every variables we declared, i.e. for all the states and clocks. As in the initializing of the location, we use the INIT statement to initialize all the clock values.

4. Location invariant declaration Each location s_i will have an INVAR statement to express its invariant $I(s_i)$ in the form $\bigwedge_{s_i \in L}(state = s_i \rightarrow \bigwedge_{\psi \in I(s_i)} \psi)$, where $I(\psi)$ are the constraints that must be fulfilled in location s_i.

Example 3.5. *On the example in Figure 3.2 we have a timed automata TA_1 with tuple $\langle L, L_0, \Sigma, X, I, E\rangle$ we define the following :*

1. *Timed automata declaration*

   We declare the timed automata TA_1 with input variable $delta_t$ as follows :

   ```
   MODULE TA_1(delta_t)
   ```

2. *Location declaration We encode the set of locations L in NuSMV as :*

   ```
     VAR state : { Wait_for_offset, Wait_for_period, Error};
   ```

   And we initialize the location Wait_for_offset as our initial state with the following command:

   ```
     INIT state = Wait_for_offset;
   ```

3. *Clock declaration The clock x for example is declared and initialized in the following encoding:*

   ```
   VAR x : real;
   INIT x = ZERO;
   ```

4. *Location invariant declaration*

   Declaring the invariant of location I through INVAR is done in the following format:

   ```
     INVAR (state = wait_for_period) -> ( x <= O1 )
   ```

We will present the example of the other encodings inline the explanations of the following sections.

The evolution of each automata

After declaring the components, we need to specify the evolution of the automata.

1. Transitions

   Each switch E_i is encoded as a local variable *transition* whose value is the enumeration of all possible transition labels with the addition of time elapse and stutter transitions. The detail of each switch is specified in a TRANS statement that expresses implication constraint in the form

$$\bigwedge_{T=\langle s_i,a,\varphi,\lambda,s_j\rangle;T\in E_i} T \to (\overline{s_i} \wedge \overline{a} \wedge \varphi \wedge s'_j \wedge \bigwedge_{x\in\lambda}(x'=z') \wedge \bigwedge_{!(x\in\lambda)}(x'=x))$$

   , where the ' superscript denotes the value of the variable at the next step and z' describes the value with which x will be reset into based on the specification in λ.

   To represent all possible transitions in NuSMV we declare a variable called *transition* that takes value in the enumeration of all possible transition. The same declaration method is also used to declare all the possible location in the model. The labels are not necessarily encoded in the NuSMV for synchronization purpose: we will explain the synchronization mechanism in the next section. However, a label $a \in \Sigma$ can be added to the encoding by declaring it as boolean variable a which is associated with the appropriate transition using the DEFINE statement. For example we can declare the label $Release_1$ in TA_1 as follows :

```
DEFINE Release_1 := process_1.transition = release_offset;
```

   Then we need to define the details of the transition : its origin state, destination state, and guard. We can define its reset actions on the clock as specified by λ here too, but for the sake of the readability we will encode them later on the evolution of the variable. We encode the two classes of transitions as follows:

   - Discrete transition

     We provide here an example definition of the transition $wait\_for\_offset\_wait\_for\_period\_1$:

     ```
     TRANS (transition = wait_for_offset_wait_for_period_1) ->
     (state = wait_for_offset & ( x = offset ))
     TRANS (transition = wait_for_offset_wait_for_period_1) ->
     (next(state) = wait_for_period)
     ```

     Both TRANS definition are conjuncted. The first statement defines the initial state, and the guard while the second statement defines the destination state. For stutter transition we only need to define that the location stays the same, as follows:

     ```
       TRANS (transition = stutter) -> (next(state) = state)
     ```

   - Time elapse We specify time elapse transition as one of the possible transitions for all the automata.

```
TRANS (transition = time_elapse) -> (next(state) = state)
```

2. Values changes on each transition

   Value changes cover the reset statements on clocks as specified by λ in the timed automata definition. As an example we show the specification for value changes on one of the clock. The evolution of the clock is expressed with ASSIGN statement, and we specify them for x as follows:

```
   ASSIGN
next(x):=
 case
  (transition = time_elapse) : (x + delta_t);
  (transition = wait_for_offset_wait_for_period_1) |
  (transition = wait_for_period_wait_for_period_2) |
  (transition = wait_for_period_wait_for_offset_3) : 0.0;
  TRUE : x;
 esac;
```

   This assignment expressed in the form of case conditions describe how during the time elapse transition the clock x will be incremented with the value of $delta_t$, then it will be reset on the transitions $wait\_for\_offset\_wait\_for\_period\_1$, $wait\_for\_period\_wait\_for\_period\_2$, and $wait\_for\_period\_wait\_for\_offset\_3$. On other transitions, the clock x will retain its value.

The composition

The variables required to encode the composition are positioned in the main module. In particular, we need the additional global variable $delta_t$ which represents the amount of time units that elapsed at each transition. This variable is declared in the main module and then as input variable to each of the automata.

In the following we offer an example of the syntax used in main module declaration. For simplicity here we show a model consisting of two automata.

```
MODULE main

VAR delta_t : real;
```

It is necessary to encode $delta_t$ as a global variable since time elapse transition is synchronized on all timed automata.

Next, we need to instantiate both timed automata:

```
VAR process_1 : TA_1(delta_t);
VAR process_2 : TA_2(delta_t);
```

here we instantiate TA_1 as $process_1$ and TA_2 as $process_2$. In the main module we will refer to the transitions, clocks, and other variable of the other module by referring to its instantiated name then the object, e.g. $process_1.x$ to refer to the clock x of the module TA_1 which is instantiated to $processs_1$. To encode the composition of both automata in particular we need to encode:

1. Time elapse transitions To synchronize the time elapse for all of the automata we create the label all_time_elapse for all the time elapse transitions in each automata.

   ```
     DEFINE all_time_elapse := process_1.transition = time_elapse;
   INVAR all_time_elapse <-> process_2.transition = time_elapse;
   ```

   We specify the possible $delta_t$ values as follows:

   ```
   INVAR  all_time_elapse -> (delta_t > ZERO)
   INVAR !all_time_elapse -> (delta_t = ZERO)
   ```

   With this, the $delta_t$ value is always zero at other transitions except time elapse. And on time elapse, it could take any arbitrarily positive value.

2. The fact that each step at least one of the transitions must happen and the prohibition of all timed automata doing null transitions at one time is encoded as:

   ```
   INVAR !((process_1.transition = stutter) &
   (process_2.transition = stutter))
   ```

3. Synchronization between transitions

   Synchronization between transitions can be defined in two ways: using the definition of same label, or through the INVAR statement. Example of defining the synchronization using a label can be seen in the example we have shown on synchronizing all time elapse event. The example of synchronizing via INVAR statement can be seen below :

   ```
     INVAR ( process_1.transition = wait_for_offset_wait_for_period_1 |
   process_1.transition = wait_for_period_wait_for_period_2 )
   <-> (process_2.transition = idle_wait_for_reset_5 )
   ```

   With the above encoding we specify that the transition $wait\_for\_offset\_wait\_for\_period\_1$ is always synchronized with $idle\_wait\_for\_reset\_5$.

   We need to also specify when one transition does not synchronize with any other transition. In this case we need to specify that other automata will be stuttering during this particular transition.

   ```
   INVAR ( process_1.transition = wait_for_period_error_4 )
   <-> (process_2.transition = stutter)
   ```

   In here we specify that the transition $wait\_for\_period\_error$ of $process_1$ will be synchronized with stutter transition on $process_2$. We take this measurement because for the sake of correctness, we only allow one transition or one synchronized transitions to happen at one step in the run of the automata.

4. The clocks always have positive values

   As the clock variables are encoded as rational variables, we need to specifically define that these variables will only have value greater than or equal to zero.

```
  INVAR process_1.x >= ZERO
& process_1.y >= ZERO
```

5. The prohibition of two consequent time elapse transitions.

```
  TRANS (process_1.transition = time_elapse) ->
   ( next(process_1.transition) != time_elapse )
```

The required property

The formula and property to be checked is expressed in LTL format. In the case of verification of schedulability the property that is going to be checked is reachability of an error state. The LTL formula of which is $\mathbf{F}(\mathit{state} = \mathit{error})$. This is expressed in NuSMV as follows :

```
INVARSPEC (process_1.state != error);
```

With this formula included the formulation of the problem is complete and it can be used as input for NuSMV. When the problem resulted in SAT, which means an error location is reached, a counterexample will be provided. This counterexample will be a path which show steps leading to error location.

Example 3.6. *Using the following assignment* $\{T_1 = 4, O_1 = 2, Timeout = 6, Threshold = 7\}$ *on the automata* TA*, the product of timed automaton* TA_1 *in Figure 3.2 and timed automaton* TA_2 *in Figure 3.3, we could run into the error state that falsifies our property as in the following run:*

$$((\mathtt{WFO}, \mathtt{Idle}), \{x = 0, y = 0, x_1 = 0\}) \xrightarrow{\delta_2} ((\mathtt{WFO}, \mathtt{Idle}), \{x = 2, y = 2, x_1 = 2\}) \xrightarrow{\mathtt{Release}_1}$$
$$((\mathtt{WFP}, \mathtt{WFR}), \{x = 0, y = 2, x_1 = 0\}) \xrightarrow{\delta_4} ((\mathtt{WFP}, \mathtt{WFR}), \{x = 4, y = 6, x_1 = 4\}) \xrightarrow{\mathtt{Abort}}$$
$$((\mathtt{Error}, \mathtt{WFR}), \{x = 4, y = 6, x_1 = 4\})$$

3.3 Formal verification for task scheduling feasibility

Wang Yi [28, 29, 45]proposed the use of an extension of timed automata to represent the activation pattern of the tasks with more flexibility compared to the classical approaches.

A timed automata is connected with tasks to represent their activation and execution time. The schedulability problem requires an extended model where subtraction of clock values is allowed. The *timed automata with subtraction* that is used in this work is an extension of the classical timed automata that we have presented before. This timed automata allows subtraction operation to the clocks to extend the reset operation in classical timed automata that only allows reset operation to zero value. The reachability problem on the timed automata with subtraction has been proven decidable [29].

Timing parameters and the hard time constraints for each task *i* are specified through the properties of tasks which are the maximum computation time C_i and the deadline D_i of each task. The encoding requires:

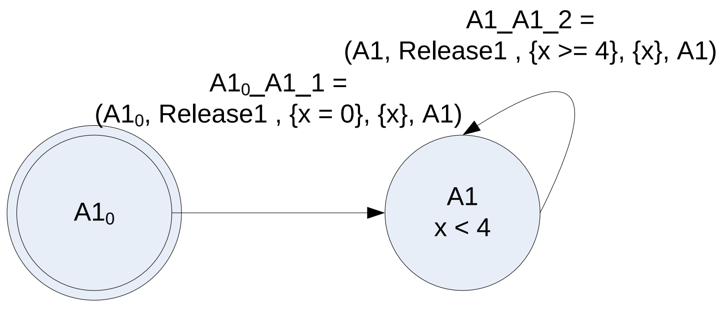

A1_A1_2 =
(A1, Release1 , {x >= 4}, {x}, A1)
A1₀_A1_1 =
(A1₀, Release1 , {x = 0}, {x}, A1)
A1₀
A1
x < 4

Figure 3.4: An example of a periodic task activation automata

1. The task activation automata
2. The scheduler automata

Both are in the form of timed automata with subtraction and will be described using the notation of the timed automata given earlier.

3.3.1 The Task Activation Automata

This first component of problem modelling in Wang Yi works models the activation pattern of tasks associated with it. Each timed automaton can represent the activation pattern of more than one task. Therefore for a task set S there can be one or more timed automaton describing the activation pattern of all of the task in S.

As described in [28] the formalism of this extended timed automata consists of the formalization of a timed automaton A_i and a mapping function $M : L \hookrightarrow \mathcal{S}$, M associating locations with tasks. Not all locations in the timed automaton are necessarily associated with one task. On the contrary, it can be also the case that one location is associated with the activation of more than one tasks. For each task τ_i related to the timed automata, an event $Release_i$ will be defined in A_i to mark its release. This event will synchronize every task activation automata with the scheduler of all tasks which will be defined later.

To describe the modelling of this task activation automata refer to Figure 3.4. It describes a simple periodic task activation where only one location is needed. In this case, we have only one location which obviously coincides with the initial location L_0. A self transition switch will be taken once every period. The automaton generates an environment event which triggers the task activation which is associated with the switch, denoted as $Release_1$. A clock x_1 is used to generate the event when a period expires. Therefore the invariant of the location associated with τ_i will only consist of the comparison of the clock with the task period, i.e. the constraint $x <= T_i$ where the period in this example is four. The guard of the switch is in the form of the condition that the clock value of the task is equal to its period, i.e the condition $x_i = T_i$. While the clock reset action of it is $x_i = 0$.

3.3.2 The Scheduler Automata

This second component of problem modelling is used to control the scheduling of the generated tasks. The desired scheduling policy for tasks scheduling is encoded in this automata.

Wang Yi and his coworkers have proposed alternative schemes to encode a non-preemptive scheduler [45] and a preemptive scheme [29]. In particular we will use the one proposed in [28] which applies to fixed priority preemptive scheduling. The process of verification is done in sequence for each task τ_i based on their priority order. The task system S is schedulable when all tasks $\tau_1, \tau_2, \ldots, \tau_n$ are verified to be schedulable. This encoding is very efficient for our problem since we do not need to represent queue as in the other schedulers, and it only requires two clock to model the scheduler automata.

The following variables are what needed by the scheduler to verify a task:

1. **c**: a clock keeping track of the time since first task to be considered (either with higher priority or equal to the task) is released.
2. **d**: a clock measuring the time since the particular checked task job is released
3. **r**: a variable which counting the time needed to finish executions of all task considered.

The locations needed in the scheduler E_i to verify the schedulability of each task τ_i are:

1. Idle

   The location where the processor is idle. It is the initial location of the scheduler and will be entered again when the job of the checked task τ_i has finished its executions or when the processor is back to idle after other jobs with higher priority for this particular task have finished their executions.

2. Busy

   The location where the processor is considered busy for the checked task job $J_{i,j}$. In this location the execution of task jobs with higher priority or other jobs $J_{i,1}, J_{i,2}, \ldots$ which is not the one being checked $J_{i,j}$ is considered.

3. Check

   This is where one particular job $J_{i,j}$ of the task τ_i is being actually checked to verify whether it can be executed within its deadline or not. The selection of which job to be checked is done non-deterministically. In a case of the task having two active jobs, the scheduler path can either go to the busy location and then to the check location, or go to the check location then the next job is being ignored

   When a job of a higher priority task is released the preemption is considered by updating the **r** variable. This way we need not represent the queue of the job.

4. Error

   The scheduler will reach this location in the case of the task job $J_{i,j}$ being checked can not finish its execution within the deadline. The reachability problem to this location is the key to verify the tasks schedulability.

To summarise, the scheduler automaton E_i is the tuple $\langle L, L_0, \Sigma, X, I, E\rangle$ (Refer to the figure 3.5 for a complete scheduler depiction):

1. $L_i = \{Idle_i, Busy_i, Check_i, Error_i\}$
2. $L_0 = \{Idle_i\}$
3. Σ= $\{Release_i, ..., Release_j | i..j \in \mathcal{P}\}$

4. $X_i = \{c_i, d_i\}$
5. $I_i = \{c_i \leq r_i, d_i \leq D_i\}$
6. $E_i = \{T_{(Idle_i, Busy_i)}, T_{(Idle_i, Check_i)}, \dots\}$

After encoding the problem into the scheduler and the task activation timed automaton as above then a reachability problem will be solved on the product of both automata. This has to be done iteratively from task τ_1 to τ_n on priority order to decide whether the task system S are schedulable or not.

Using timed automata with subtraction, the decidability is guaranteed if the clock values are bounded with certain maximal constant. Indeed, such a bound results in a finite number of states equivalence classes. The state space then can be partitioned into a finite number of equivalence classes. The use of Difference Bound Matrices (DBM) technique allows solving the problem, thus proving that it is decidable [29]. Hence, the scheduler uses some maximum value bound variables, namely C_{Max} and R_{Max} for the clocks. When the bound C_{Max} is reached, the clock value is reset. The optimum values of C_{Max} and R_{Max} is obtained through iteration.

In NuSMV this kind of timed automata extension can be straightforwardly encoded the same way we encode the classical timed automata. This is because the clocks are represented as real variables in NuSMV. Thus, we can perform any linear arithmetical operation to the clocks, including the subtraction operation and the reset to zero value. In addition, the limitation of the clocks to a maximum value is not required.

Figure 3.5 shows the example of the modelling of two task scheduling. The switches are written in the switch notation of timed automata, $\langle s_i, a, \varphi, \lambda, s_j \rangle$ (Refer to section 3.2.1). Task 1 has a higher priority than task 2 and both are periodic tasks with the period of 4 and 16 and deadlines equal with their periods. The variable C_1 and C_2 represent the maximum computation time for task 1 and 2.

The variable $\mathbf{C}^{max}$ gives the maximum clock (**c**) value limit. As clock (c) retains the amount of computation time that has been done, $\mathbf{C}^{max}$ ensures that its value will not exceed $\mathbf{C}^{max}$. When **c** has reached the value $\mathbf{C}^{max}$, it will be reduced by $\mathbf{C}^{max}$ to zero, thus ensuring the finite state space of the system. Related to variable **r** value is the bound variable $\mathbf{R2}^{max}$. Variable **r** remembers the remaining computation time that must be done. $\mathbf{R2}^{max}$ gives the maximum limit where **r** value is still allowed for the schedulability of the scheduler. Upon exceeding its value, the system then will immediately move to an error state. Since it is already known before the subsequent deadline is reached that there will be a violation.

3.3.3 NuSMV implementation of timed automata verification

As in NuSMV the limitation on **r** and **c** are not necessary, the scheduler automata can be simplified, i.e. the transitions for the purpose of enforcing the value limit can be omitted. The resulting NuSMV encoding of scheduler automata in the figure 3.5 the transition $Check\_Check\_10$, $Check\_Error\_11$, $Busy\_Busy\_16$, and $Busy\_Error\_17$ are omitted.

Example 3.7. *We present an example run of the two activation automata and scheduler of the second task depicted in Figure 3.5 encoded in NuSMV. Using the following assignment* $\{C1 = 3.5, C2 = 1, T1 = D1 = 4, T2 = D2 = 6\}$*, then one of the possible run is*

$$((\texttt{A1}_0, \texttt{A2}_0, \texttt{Idle}), \{x=0, y=0, c=0, d=0\}, r=0) \xrightarrow{\texttt{Release}_1} ((\texttt{A1}, \texttt{A2}_0, \texttt{Busy}), \{x=0, y=0, c=0, d=0\}, r=3.2) \xrightarrow{\texttt{Release}_2}$$
$$((\texttt{A1}, \texttt{A2}, \texttt{Check}), \{x=0, y=0, c=0, d=0\}, r=4.2) \xrightarrow{\delta_4} ((\texttt{A1}, \texttt{A2}, \texttt{Check}), \{x=4, y=4, c=4, d=4\}, r=4.2) \xrightarrow{\texttt{Release}_1}$$
$$((\texttt{A1}, \texttt{A2}, \texttt{Check}), \{x=0, y=4, c=4, d=4\}, r=7.4) \xrightarrow{\delta_2} ((\texttt{A1}, \texttt{A2}, \texttt{Check}), \{x=2, y=6, c=6, d=6\}, r=7.4) \rightarrow$$
$$((\texttt{A1}, \texttt{A2}, \texttt{Error}), \{x=2, y=6, c=6, d=6\}, r=7.4)$$

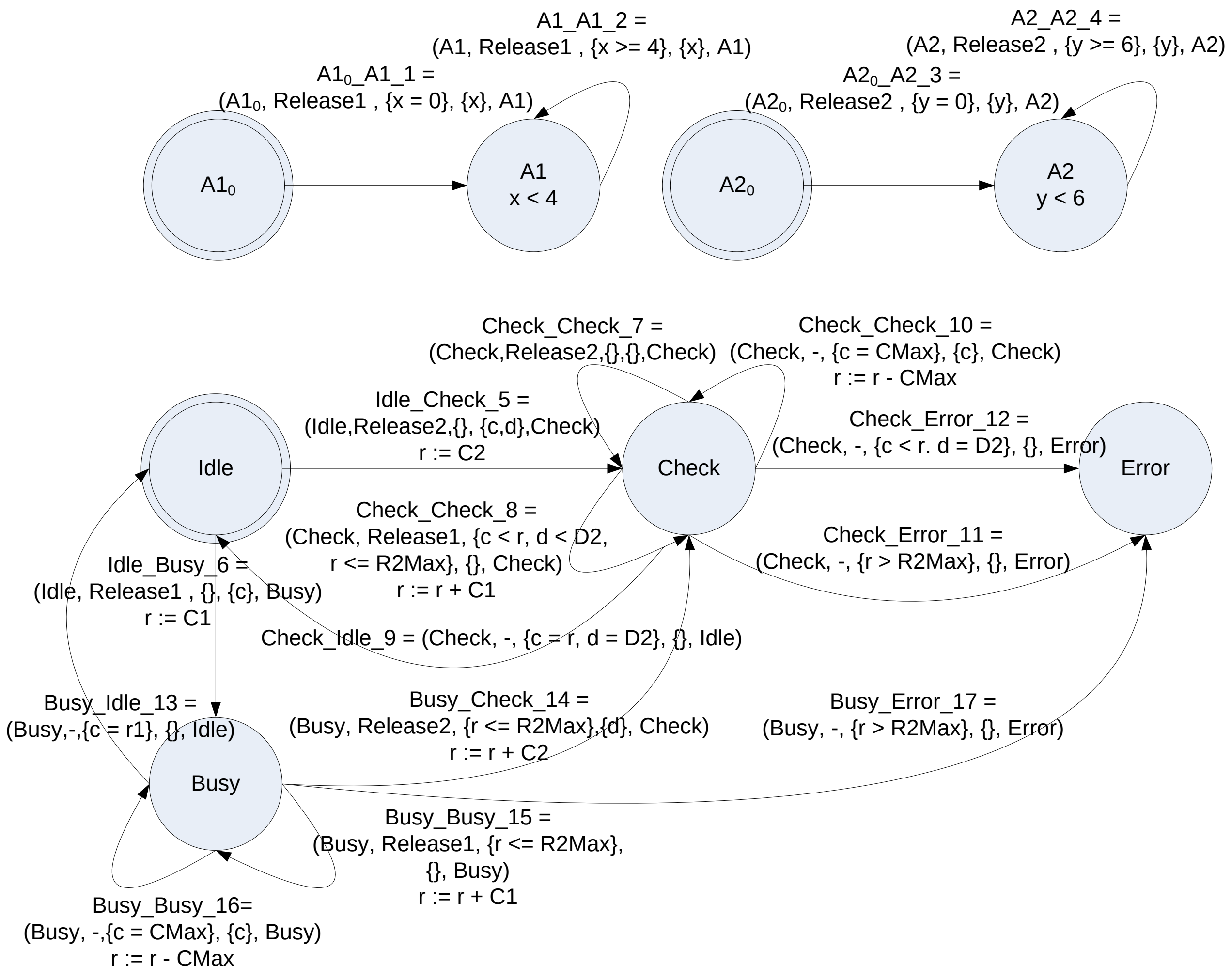

A1_A1_2 =
(A1, Release1 , {x >= 4}, {x}, A1)
A1₀_A1_1 =
(A1₀, Release1 , {x = 0}, {x}, A1)
A1₀
A1
x < 4
A2_A2_4 =
(A2, Release2 , {y >= 6}, {y}, A2)
A2₀_A2_3 =
(A2₀, Release2 , {y = 0}, {y}, A2)
A2₀
A2
y < 6
Check_Check_7 =
(Check,Release2,{},{},Check)
Check_Check_10 =
(Check, -, {c = CMax}, {c}, Check)
r := r - CMax
Idle_Check_5 =
(Idle,Release2,{}, {c,d},Check)
r := C2
Check_Error_12 =
(Check, -, {c < r. d = D2}, {}, Error)
Idle
Check
Error
Check_Check_8 =
(Check, Release1, {c < r, d < D2,
r <= R2Max}, {}, Check)
r := r + C1
Check_Error_11 =
(Check, -, {r > R2Max}, {}, Error)
Idle_Busy_6 =
(Idle, Release1 , {}, {c}, Busy)
r := C1
Check_Idle_9 = (Check, -, {c = r, d = D2}, {}, Idle)
Busy_Idle_13 =
(Busy,-,{c = r1}, {}, Idle)
Busy_Check_14 =
(Busy, Release2, {r <= R2Max},{d}, Check)
r := r + C2
Busy_Error_17 =
(Busy, -, {r > R2Max}, {}, Error)
Busy
Busy_Busy_15 =
(Busy, Release1, {r <= R2Max},
{}, Busy)
r := r + C1
Busy_Busy_16=
(Busy, -,{c = CMax}, {c}, Busy)
r := r - CMax

Figure 3.5: The example of task activation and scheduler automata for 2 tasks

At initial state all automata are in their initial locations $(A1_0, A2_0, Idle)$ *all clocks* $\{x, y, c, d\}$ *are zero and the variable* r *counting the time needed to finish executions of all task is zero. Both tasks* τ_1 *and* τ_2 *are immediately released. First, transition* $\texttt{A1}_0\texttt{\_A1\_1}$ *synchronizing with transition* `Idle_Busy_6` *on label* $\texttt{Release}_1$ *takes the timed automata to state* $(\texttt{A1}, \texttt{A2}_0, \texttt{Busy})$*, reset the clock* c *and* x *and assign* $C1$ *value to* r*. Then, transition* $\texttt{A2}_0\texttt{\_A2\_3}$ *synchronizing with transition* `Busy_Check_14` *on label* $\texttt{Release}_2$ *moves the timed automata to state* $(\texttt{A1}, \texttt{A2}, \texttt{Check})$*, reset the clock* d *and increment the value of* r *with* $C2$*. At this point the job of task* τ_2 *that we are checking is about to be executed, pending the execution of job* τ_1*. A* δ*-transition of 4 time unit then taken, increasing all clocks value by 4. The switch* `A1_A1_2` *is then enabled and taken in synchronization with transition* `Check_Check_8` *on* $Release_1$ *label. This increases the value of* r *with* $C1$*. With the next time elapse,* δ*-transition of 2 time unit we then encounter the deadline* $D2$ *of task* τ_2*,* $d = D2$*. At this point the value of clock* c *is less than the remaining execution time* r*, which means the job of task* τ_2 *has not finished being executed even though the deadline time is reached. As all the guards* $c < r \wedge d = D2$ *of the switch* `Check_Error_12` *are satisfied, the switch is taken and we reached error state. Thus, the task* τ_2 *is not schedulable.*

3.3.4 Sensitivity Analysis

The approach describerd above allows us to solve the Problem 1 in the *ground sub-case*, meaning that all parameters have a fixed value. However, it is not evident how to tackle Problem 2, where parameters are allowed to vary. In the next chapters, we will show how the model used Wang Yi and co-workers [28], which is particularly efficient for it uses only two clocks, can be adapted to the study of parametric systems.

Chapter 4

Real–Time System Sensitivity Analysis on Parametric Timed Automata

4.1 Parametric Timed Automata

Parametric Timed Automata is an extension to the classical Timed Automata that we proposed to model parametric real–timesystem. Let V be a set of symbols, with valuations over the rationals. A term over symbols in V is defined as $a_1 \cdot v_1 + \ldots + a_n \cdot v_n$, with $v_i \in V$ and $a_i \in \mathbb{Q}$. A constraint over V is defined as $t \circ k$, where t is a term over V, $k \in \mathbb{Q}$, and $\circ \in \{<, \leq, >, \geq, =, \neq\}$. We denote the set of terms and constraints over V as $\mathcal{T}(V)$ and $\mathcal{C}(V)$, respectively. $\mathcal{B}(V)$ is the set of boolean combinations of constraints over V, with the standard connectives $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation). We denote with $\mathcal{U}(V)$ the set of update statements of the form $v := t$, with $v \in V$, and $t \in \mathcal{T}(V)$. An assignment over a set of variables V is a function from V to $\mathbb{Q}$. We use μ_V to denote assignments to variables in V. The evaluation of a constraint c on an assignment μ_V returns the boolean value true when constraint satisfied, and false when violated. We do not distinguish between a formula $\varphi \in \mathcal{B}(V)$, the set of satisfying assignments over V, and the corresponding region of the space $\mathbb{Q}^{|V|}$. We write $\mu_V \models \varphi$ to state that μ_V belongs to the region φ. We notice that the conjunction of two formulas amounts to the intersection of the corresponding regions, negation to complementation, and union to disjunction.

A *parametric timed automaton* is a tuple $\langle L, L_0, \Sigma, X, P, \Gamma, I, E \rangle$, where

- L is a finite set of locations
- $L_0 \subseteq L$ is the set of initial locations
- Σ a finite set of labels
- X is a finite set of variables
- P is a finite set of parameters
- $\Gamma \subseteq \mathcal{B}(P)$ is the parameter space
- $I : L \to 2^{\mathcal{C}(X \cup P)}$ is the invariant map
- $E \subseteq L \times \Sigma \times \mathcal{C}(X \cup P) \times 2^{\mathcal{U}(X)} \times L$ is the set of switches.

As in the classiscal notion of timed automata we presented in previous chapter, a switch $\langle l, a, \varphi, \lambda, l' \rangle$ represents a transition from location l to location l', in response to $a \in \Sigma$, when guard φ holds; the value of the variables in X is updated according to the function $\lambda : X \times P \to X$. The set X is the disjoint union of X_c (clocks) and X_s (state variables). The value of clocks can either be reset on each transitions,

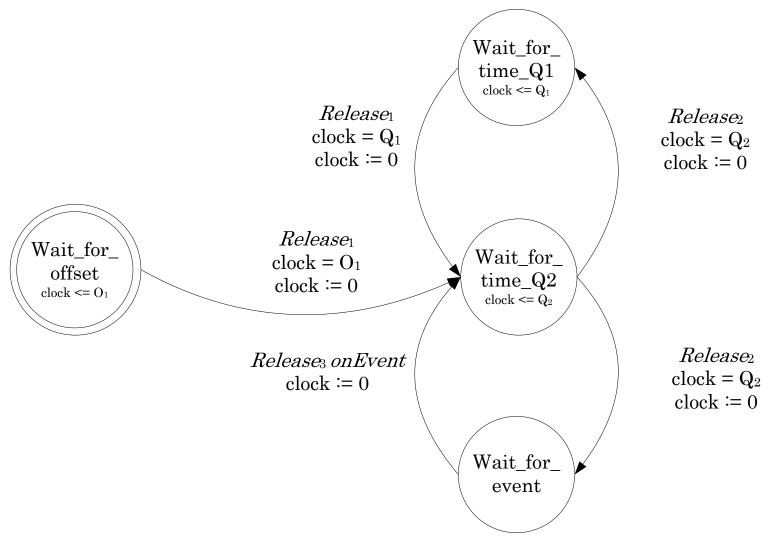

Wait_for_ time_Q1 clock <= Q1
Release1 clock = Q1 clock := 0
Release2 clock = Q2 clock := 0
Wait_for_ offset clock <= O1
Release1 clock = O1 clock := 0
Wait_for_ time_Q2 clock <= Q2
Release3 onEvent clock := 0
Release2 clock = Q2 clock := 0
Wait_for_ event

Figure 4.1: Example of parametric timed automata: tasks activation

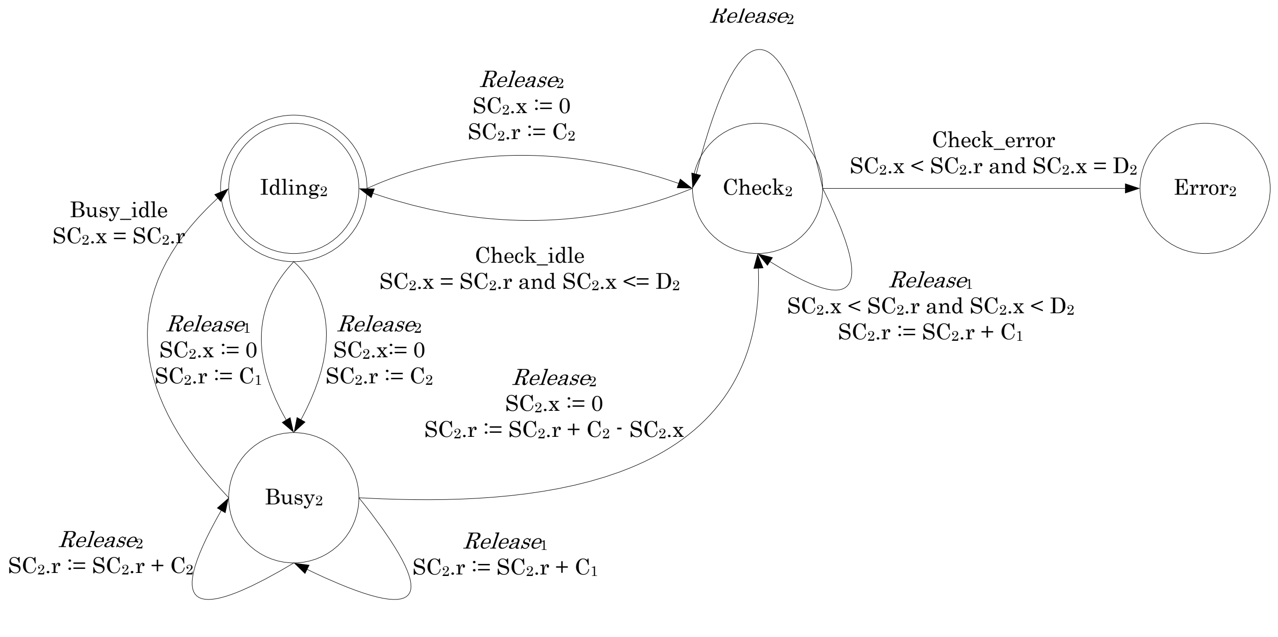

Release2
Release2 SC2.x := 0 SC2.r := C2
Check_error SC2.x < SC2.r and SC2.x = D2
Idling2
Check2
Error2
Busy_idle SC2.x = SC2.r
Check_idle SC2.x = SC2.r and SC2.x <= D2
Release1 SC2.x < SC2.r and SC2.x < D2 SC2.r := SC2.r + C1
Release1 SC2.x := 0 SC2.r := C1
Release2 SC2.x:= 0 SC2.r := C2
Release2 SC2.x := 0 SC2.r := SC2.r + C2 - SC2.x
Busy2
Release2 SC2.r := SC2.r + C2
Release1 SC2.r := SC2.r + C1

Figure 4.2: Example of parametric timed automata: schedulability checker automata for task τ_2 (SC_2) (right)

or it can grow linearly in time in each location. The value of state variables can be changed only as a result of a reset action when a transition is taken.

Example 4.1. *Examples of parametric timed automata are given in Figure 4.1 and Figure 4.2. In the timed automaton in Figure 4.1, L contains four locations,* `Wait_for_offset`*,* `Wait_for_time_Q1`*,* `Wait_for_time_Q2`*, and* `Wait_for_event`*, referred to as* `WFO`*,* `WFQ1`*,* `WFQ2`*, and* `WFE` *in the rest of the text. The initial location L_0 is* `WFO`*. The labels are* $\mathtt{Release}_1$*,* $\mathtt{Release}_2$*, and* $\mathtt{Release}_3\mathtt{onEvent}$*. There is one clock variable (clock), three parameters (Q_1, Q_2, O_1). An example of invariant condition is the constraint $clock \leq O_1$ associated with location* `WFO`*; an example of switch is $\langle$*`WFO`*,$\mathtt{Release}_1$, $clock = O_1, clock := 0,$* `WFQ2`*$\rangle$, which represents a transition from* `WFO` *to* `WFQ2`*. The transition can be taken when $clock = O_1$, and it resets clock to 0. The automaton in Figure 4.2 has a state variable $SC_2.r$ that is reset, for instance, on the transition from* $\mathtt{Busy}_2$ *to* $\mathtt{Check}_2$*.*

In PTA, a state is (l, μ_P, μ_X), with $l \in L$, μ_P assignment to the parameters such that $\mu_P \models \Gamma$, and μ_X assignment to the variables such that $\mu_X \cup \mu_P \models I(l)$; it is initial iff $l \in L_0$. Let σ be a switch $\langle l, a, \varphi, \lambda, l' \rangle \in E$. Then we say that a state s reaches s' through σ, written $s \xrightarrow{\sigma} s'$, iff $\mu_P = \mu'_P$, $\mu_X \models \varphi$, and $\mu'_X := \lambda(\mu_X)$ where λ is a reset function to the selected clocks. We say that a state s reaches a state s' through δ_t iff $l = l'$, $\mu_P = \mu'_P$, and $\mu'_{X_c} := \mu_{X_c} + t$.

A run of length k of a PTA is a sequence of states $(l^0, \mu_P, \mu_X^0), (l^1, \mu_P, \mu_X^1), \ldots, (l^k, \mu_P, \mu_X^k)$ iff the first state is initial, and for $i = 0, \ldots, k-1$ it holds that $(l^i, \mu_P, \mu_X^i) \xrightarrow{\sigma} (l^{i+1}, \mu_P, \mu_X^{i+1})$ for some switch σ, or $l^i = l^{i+1}$ and $(l^i, \mu_P, \mu_X^i) \xrightarrow{\delta_t} (l^{i+1}, \mu_P, \mu_X^{i+1})$. The presence of the parameter valuation μ_P marks a difference w.r.t. the timed automata run presented in the previous chapter. It is worth to observe that parameters are assigned a value in their space that is retained throughout the run.

Example 4.2. *If μ is $\{O_1 = 4, Q_2 = 20\}$, then a run for the automaton in Figure 4.1 is*

$$\begin{array}{llll}
(\mathtt{WFO}, \mu, \{clock = 0\}) & \xrightarrow{\delta_4} & (\mathtt{WFO}, \mu, \{clock = 4\}) & \xrightarrow{\mathtt{Release}_1} \\
(\mathtt{WFQ2}, \mu, \{clock = 0\}) & \xrightarrow{\delta_{20}} & (\mathtt{WFQ2}, \mu, \{clock = 20\}) & \xrightarrow{\mathtt{Release}_2} \\
(\mathtt{WFE}, \mu, \{clock = 0\}) & & &
\end{array}$$

We rely on the same notion of composition [1], that we presented in previous chapter, between automata having disjoint sets of variables. The locations of the composite automaton are given by the cartesian product of the locations of the two automata; the same applies to the initial locations. Parameters and variables of the composite are the union of the parameters and variables of the components. The switches of the composite are given by the combination of the switches of the components.

Example 4.3. *Consider, as an example, the composition of the two automata in Figure 4.1 and 4.2. A possible run associated to the assignment of parameters $\{O_1 = 4, Q_1 = 15, Q_2 = 10, C_1 = 20, C_2 = 5, D_2 = 12\}$ is reported below.*

Step	*Location*	$(A.clock, SC_2.x, SC_2.r)$	*Transition*
1	(*WFO*, *Idling*)	(0, 0, 0)	(δ_4)
2	(*WFO*, *Idling*)	(4, 4, 0)	$(\mathtt{Release}_1)$
3	(*WFQ2*, *Busy*)	(0, 0, 20)	(δ_{10})
4	(*WFQ2*, *Busy*)	(10, 10, 20)	$(\mathtt{Release}_2)$
5	(*WFQ1*, *Check*)	(0, 0, 15)	(δ_{12})
6	(*WFQ1*, *Check*)	(12, 12, 15)	$(\mathtt{A.Stut}, \mathtt{SC}_2\mathtt{.Check\_error})$
7	(*WFQ1*, *Error*)	(12, 12, 15)	–

During the first step, the composite automata evolves with a δ transition of duration 4, and the clocks $A.clock$ and $SC_2.x$ are incremented of the same amount. The second step corresponds to a switch associated with the label $\texttt{Release}_1$. *This switch is the result of the composition of two switches in each component automaton: from* `Idling` *to* $\texttt{Busy}_2$ *for the automaton on the right, and from* `WFO` *to* `WFQ2` *for the automaton on the left. The guards of both switches have to evaluate to true simultaneously and their reset actions are combined. Similarly for the other transitions. At step 6, automaton* `A` *stutters, while* $\texttt{SC}_2$ *carries out a transition labelled by* `Check_error`.

The framework described above is very general, and captures several special cases of interest. With respect to the traditional model of timed automata [1], our system differs in the following main points :

1. The presence of parameter

   This will allow us to analyse the timed automata parametrically as we will describe in the next chapter.

2. The presence of auxiliary variables

   Instead of having only clocks in each timed automata, we allow the existence of state variables in X_s. This allow other variables that might be needed for the system, e.g. buffer system variables, counter, to be represented in PTA.

3. Generality of the update statements.

   We allow arbitrary linear expression operations to be performed on the clocks. While for the "classic" timed automata clocks can only be reset ($x := 0$), in our setting an update may have a linear polynomial in the right hand side of the assignment. We can obtain standard timed automata if the set of parameters and of state variables are empty, and the update constraints have the form $x := 0$. If update constraints have the form $x := x - c$, with $c \in \mathbb{Q}$, we obtain the TA with subtraction [28].

4.2 Modelling real–time system in PTA

In order to represent parametric real-time systems, we can combine the ideas introduced in the previous chapters with the notion of PTA. Indeed, the activation of jobs can be modelled by a parametric timed automaton, where activation events are associated with transition labels. As an example, consider Figure 4.1.A. The timed automaton activates tasks τ_1, τ_2, and τ_3 in correspondence of the labels $\texttt{Release}_1$, $\texttt{Release}_2$ and $\texttt{Release}_3\texttt{onEvent}$. Jobs of task τ_1 are activated after a certain fixed time Q_1 has elapsed (and similarly for τ_2); a job of task τ_3 is nondeterministically activated. The example shows that very complex activation patterns can be captured by using timers, parameters, and external events.

The traditional scenarios of the real-time literature are contained in our framework. For instance, if S is a periodic task system, the activation automaton simply consists of a network of n independent timed automata, each representing the activation of periodic task τ_i, as shown in Figure 4.3.1; each task τ_i has a period T_i, a fixed duration of time between two activation events, and it may have an offset O_i for its first activation time. Its activation automaton contains a clock, which is reset every time the period is reached and $\texttt{Release}_i$ is fired.

The parameters in a real-time task system as defined in Definition 2.3 made up the set of parameters P in PTA. On periodic task set for example we can have our parameters $\boldsymbol{P} = \{(C_1, D_1), \ldots, (C_n, D_n),\}$ associated with each task.

Each design parameter in $\boldsymbol{P}$ can have a fixed value or be a free parameter. As we discussed in Chapter 2 very often, the free parameters are a subset of the ones listed above. For instance, the period T_i is typically imposed by external constraints (e.g., on the sampling rate of the sensors). Computation times can be considered as free parameters in as much as the designer can refine the implementation to comply with the constraints dictated by the platform. Other parameters, such as the offsets, can be selected with a certain freedom. As a notational convention we will use bold fonts for quantities that are not allowed to change. For instance, if in a task set we fix the period, we will denote it by $\mathbf{T}_i$.

As a subcase of our parametrized task systems, we can define the ground sub-case.

Definition 4.1. A ground sub-case for parametrized task systems is a task system $\mathcal{S} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ associated with a vector of design parameters $\boldsymbol{P}$ where each design parameter in $\boldsymbol{P}$ is assigned a fixed value, i.e., there is no free parameter in the task system.

The objective of our work is to obtain schedulability region as presented in Problem 2 we defined in Chapter 2 and we present our approach based on PTA.

4.3 Sensitivity analysis using PTA

The objective of this paper is to find the largest region in the parameter space for which a task set respects some temporal properties as we defined in Problem 2In contrast to the ground sub-case, identifying the feasibility region is a relatively new problem. As reported in Section **??**, one of the few papers that handles parametric task set producing the schedulability region is [11], in which the attention is restricted to periodic tasks with zero offsets. We describe here an approach for producing the feasibility regions, constructed as a union of convex polyhedra, for general task sets. The method applies to all properties whose violation can be modelled as reachability of an error state. In the last chapter, we showed how this applies to the schedulability problem, and, in the following, we will still restrict our attention to this problem. The generalisation to different properties simply amounts to the introduction of different error states.

4.3.1 Parametric Verification of Temporal Properties (PTVP) algorithm

Our approach to computing the feasibility region of a parametrized task system relies on the generalization of the approach presented in [28] to the parametrized case. Within this framework, we propose a novel algorithm, Parametric Verification of Temporal Properties (PTVP) algorithm, for the symbolic computation of the feasibility region. The pseudo-code is shown in Figure 4.1. As in [28], we iterate over the tasks in the system $\mathcal{S}$, starting from the task with highest priority ($i = 1$). For each of these we compute the schedulability sub-region $\mathcal{R}_i$ (i.e., the region of parameters such that task τ_i is schedulable).

The computation of the schedulability sub-region for each task is done in the inner loop by enumerating each trace to the error state (the trace hunting step can be made in different ways), and then generalizing by extracting the region of parameters that preserve the validity of the trace. The "numeric" timed automata (used as schedulability checkers and to generate the activation events) are replaced with parametric timed automata. For instance, in the activation automata shown in Figure 4.1.(A) the parameters O_1, Q_1 and Q_2 can be free parameters instead of numeric values. The identifier `PTA` denotes the parametric automata used to construct the schedulability sub-region for each task. Every time we enter into the inner loop, `PTA` is initialized with the Schedulability Automata Network (SAN) SAN_i utilized for task τ_i, given by the composition of the activation automaton and of the schedulability checkers SC_i. We construct the region $\mathcal{R}_i$ by a sequence of iterations of the inner loop, as detailed below.

```
Require: PTA describing activations and scheduling of n tasks
Ensure: Schedulability Region
 1: for $i = 1$ to n do
 2:    PTA.init(ParamSchedProblemForTask(i))
 3:    j = 0
 4:    while PTA.reachable(Error) do
 5:       trace = PTA.get_trace()
 6:       Unfeasible[j] = PTA.get_parameter(trace)
 7:       PTA.add_constraints( negate( Unfeasible[j]))
 8:       j++
 9:    Feasible[i] = not(big_or(0, j, Unfeasible))
10: Return big_and(0, n, Feasible)
```

Algorithm 4.1: Parametric Verification of Temporal Properties (PTVP) algorithm: Iterative algorithm for PTA schedulability region analysis

Each iteration starts by calling the function `PTA.reachable`. This is a call to the model checker to see whether the error state is reachable. Three things can happen: reachable, not reachable, or failure to compute an answer in the given resource bounds. The algorithm is not guaranteed to terminate, and it is possible that a trace to the error state (henceforth referred to as *error trace*) does not exist, but the algorithm cannot find out this result in finite time. In the next subsection, we will show decidability at least for a very important special case (periodic tasks with free offsets), while further investigations on the convergence in more general cases are reserved for future work.

Suppose that the j iteration of the inner loop for task τ_i `PTA.reachable` produces reachable, i.e., the model checking problem $SAN_i^j \models EF scheduler.error$. The call to the function `PTA.getTrace()` will then supply us with a trace π_i^j. The next step, carried out by the `PTA.get_parameter` function, is to identify the region of parameters ψ_i^j that make the trace valid (in the spirit of [26]). As shown in the next section, the operation requires a symbolic manipulation of the trace and a projection operation. The region ψ_i^j is a convex polyhedron, identified by a disjunction of linear constraints over the parameters $\boldsymbol{P}$, and is a subset of the complement of $\mathcal{R}_i$. We use the information collected in this way as as an additional constraint for the next iteration. Namely, we limit the search space of $SAN_i^{(j+1)}$ by `PTA.add_constraints`, by inserting, as an additional constraint to the reachability problem, the conjunction of $\neg(\psi_i^j)$ with the constraints $\neg(\psi_i^1) \wedge \neg(\psi_i^2) \wedge \ldots \wedge \neg(\psi_i^{j-2}) \wedge \neg(\psi_i^{j-1})$ obtained in the previous iterations.

When the function `PTA.reachable` produces unreachable, we can compute the feasibility region $\mathcal{R}_i$ as $\mathcal{R}_i = \neg(\psi_i^1 \vee \psi_i^2 \ldots \vee \psi_i^J)$ where J is the maximum j, after which the Error state is unreachable.

The total schedulability regions is the intersections of the schedulability regions found for each task: $\mathcal{R} = \cap_{i=1}^n \mathcal{R}_i$, where n is the number of tasks.

Example 4.4. *Consider a system of two periodic tasks $S = \{\tau_1, \tau_2\}$ with deadlines and periods given by $D_1 = 7, T_1 = 10$ and $D_2 = 6, T_2 = 10$.*

The activation automata for this case is a network of two independent automata. The activation automata of task τ_1 is shown in Figure 4.3.1. The schedulability checker for task τ_2, SC_2, is exactly the one shown in Figure 4.2. As shown in the figure, for task τ_2 the only relevant transitions that affect the counter r are determined by the only task with a higher priority τ_1. The schedulability checker SC_1 for task τ_1 is simpler since τ_1 does not suffer any interference from higher priority tasks. We do not report it

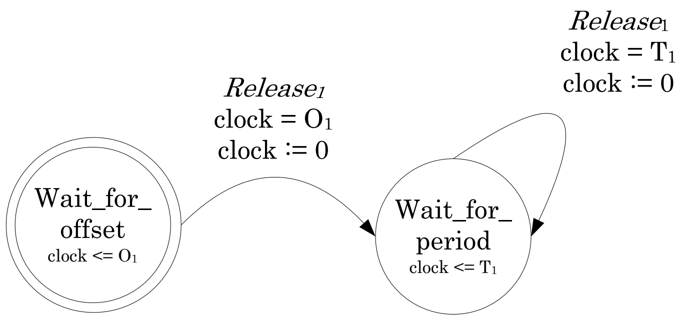


Figure 4.3: Task activation automata for periodic tasks

here for the sake of brevity.

We now illustrate the algorithm by adapting the example in C_1, C_2 and O_2 are free parameters while we set $O_1 = 0$. The computation of the feasibility region proceeds as shown in 4.4. For task τ_1, the schedulability region is found in one iteration of the inner loop and is simply given by $C_1 < 7$. The region $\mathcal{R}_2$ is the complement of the union of the polyhedra produced in six iterations. In the second column of the table, we show the restricted search space for the reachability problem. The schedulability region $\mathcal{R}$, given by the conjunction of $\mathcal{R}_1$ and $\mathcal{R}_2$, is expressed as follows:

$$C_1 + C_2 < 6 + O_2 \wedge C_1 + C_2 < 10 \wedge$$
$$C_1 + C_2 < 6 \wedge C_2 < 10 - O_2 \wedge C_1 < 7 \wedge C_2 < 6$$

The 3D plot of $\mathcal{R}$ is depicted in Figure 4.5. As shown above, some of the constraints found are redundant; this problem can be solved using classical methods for the elimination of redundant constraints. A second remark, since we consider a periodic task set, the function `PTA.reachable` *produces a negative result (Error unreachable) in finite time.*

4.3.2 Algorithm termination guarantee for periodic task sets

While the PTVP algorithm presented above can be applied to models represented as general task activation patterns, in this section we will present strong convergence results for the special case of periodic task sets.

In particular, we will make the following assumptions: 1) the period of the task $\mathbf{T}_i$ and the relative deadline $\mathbf{D}_i$ of the tasks are fixed, 2) for notational convenience we will assume that tasks are ordered by decreasing priority. Therefore, the parameter space for each task is defined as $P_i = \{C_i\}$ and $P_s = O_1 \times O_2 \ldots \times O_n$.

For this type of task set, it is possible to state the following result (see [41]):

Theorem 4.1. Consider a periodic task set and let $H = \text{lcm}\{\mathbf{T}_1, \ldots, \mathbf{T}_n\}$ and $O = \max\{\mathbf{O}_1, \ldots, \mathbf{O}_n\}$. Then the following two statements are equivalent: I) The task set is unschedulable, II) A deadline miss occurs in the time interval $[0, 2H + O]$.

As a consequence of the above we can write the following:

Lemma 1. *Consider Problem 2 and assume that $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ is the periodic task set described above and that there is an upper bound O for the maximum offset. Then the number of error traces for the* `PTA` *described above that could terminate with an error condition is finite.*

$Step$	$Constraint$	$ResultConstraint$
Task 1		
1	–	$C_1 > 7$
Task 2		
1	–	$C_1 + C_2 > 6 + O_2$
2	$\neg(C_1 + C_2 > 6 + O_2)$	$C_1 + C_2 > 10 \wedge$ $O_2 = 10 \wedge$ $2C_1 + C_2 > 6 + O_2$
3	$\neg((C_1 + C_2 > 6 + O_2) \vee$ $(C_1 + C_2 > 10 \wedge$ $O_2 = 10 \wedge$ $2C_1 + C_2 > 6 + O_2))$	$C_2 > 6$
4	$\neg((C_1 + C_2 > 6 + O_2) \vee$ $(C_1 + C_2 > 10 \wedge$ $O_2 = 10 \wedge$ $2C_1 + C_2 > 6 + O_2) \vee$ $(C_2 > 6))$	$C_1 + C_2 > 10 \wedge$ $2C_1 + C_2 > 6 + O_2$
5	$\neg((C_1 + C_2 > 6 + O_2) \vee$ $(C_1 + C_2 > 10 \wedge$ $O_2 = 10 \wedge$ $2C_1 + C_2 > 6 + O_2) \vee$ $(C_2 > 6) \vee$ $(C_1 + C_2 > 10 \wedge$ $2C_1 + C_2 > 6 + O_2))$	$C_1 + C_2 > 6 \wedge$ $O_2 = 10$
6	$\neg((C_1 + C_2 > 6 + O_2) \vee$ $(C_1 + C_2 > 10 \wedge$ $O_2 = 10 \wedge$ $2C_1 + C_2 > 6 + O_2) \vee$ $(C_2 > 6) \vee$ $(C_1 + C_2 > 10 \wedge$ $2C_1 + C_2 > 6 + O_2) \vee$ $(C_1 + C_2 > 6 \wedge$ $O_2 = 10))$	$C_2 > 10 + O_2 \wedge$ $C_1 + O_2 > 6$

Figure 4.4: Example run of the algorithm

Proof. Consider the problem of verifying the schedulability of task τ_i. We can denote a trace by the sequence of labels associated to the transitions. Clearly, error traces are given by:

$$\begin{aligned}\pi_j \ &= (\texttt{Release}_1, \ldots, \texttt{Release}_i, \texttt{Check\_idle})^+ \\ &\quad \texttt{Release}_i, \texttt{Check\_error}\end{aligned}$$

where we used the standard notation for regular expressions. Denote by $\lambda(\pi_j, x)$ the number of occurrences of symbol x inside the trace π_j. Based on Lemma 4.1, we can reduce the analysis to the events that occur in the time horizon $[0, 2H + O]$. Therefore, it is easy to see that for any error trace π_i, we have: 1) $\lambda(\pi_j, \texttt{Release}_h) \leq \left\lceil \frac{2H+O}{\mathbf{T}_i} \right\rceil$ for any $h \leq i$, 2) $\lambda(\pi_j, \texttt{Check\_idle}) \leq \lambda(\pi_i, \texttt{Release}_i)$. The number of strings composed using a finite number of symbols is clearly finite, which leads to the claim. □

As an immediate consequence of the above, we can write the following:

Lemma 2. *Consider Problem 2 and assume that $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ is the periodic task set described above. Assume that we solve the problem by using the PTVP algorithm in Figure 4.1. If 1) function* `PTA.get_trace()` *enumerates all witness traces, 2) function* `PTA_get_parameter(.)` *identifies all parameters that validate a given error trace, then the PTVP algorithm computes the feasibility region in finite time.*

This result can be used to ensure that the proposed PTVP algorithm is guaranteed to terminate in this important special case. However, the bound computed for the number of possible events in the proof of Lemma 1 is very conservative.

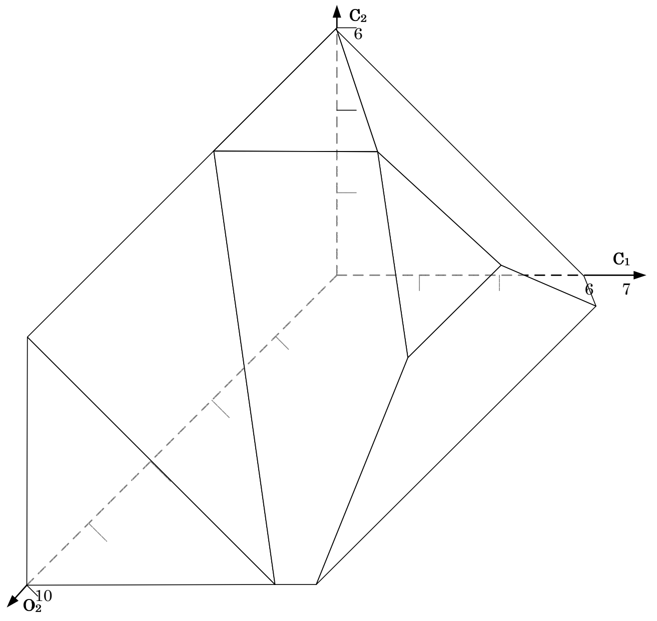


Figure 4.5: The feasibility region of the problem in 3D plot of C_1, C_2 and O_2

4.4 Symbolic implementation

In this section we discuss how to algorithmically manipulate PTA's. In fact, standard techniques for the manipulation of timed automata are typically based on numerical techniques, where suitable degrees of parameterization are not provided.

Our approach relies on a symbolic approach to model checking (timed) automata [9]. The state space of the system is represented by means of a vector of *current state* variables V: an assignment of values to the variables in V represents a state of the system. As standard in symbolic model checking, the discrete part of the automaton is encoded (e.g., transitions, labels, and locations) by means of boolean variables. Continuous (clock and state) variables and parameters are represented by means of real-valued variables. A formula such as $\texttt{Loc}_i \rightarrow x - y \leq O_1$ can be used to represent the fact that in location i (corresponding to the boolean variable $\texttt{Loc}_i$ being true), the specified condition over x, y and O_1 must hold.

In order to express the dynamics of the system, another set of variables V', called the *next state* variables, is introduced. Intuitively, a transition is represented as an assignment to the V and V' variables. For instance, $\texttt{Trans}_{i,j} \rightarrow (x - y \geq 5) \wedge (x' = 0) \wedge (y' = y)$ states that for a transition from location i to location j to take place (corresponding to $\texttt{Trans}_{i,j}$ being true), $(x - y \geq 5)$ must hold (as a precondition), the value of x after the transition will be 0, while the value of y will be unchanged. In [9], a procedure for compositional symbolic representation of timed automata is described. It can be generalized to the case of PTA's by considering that each parameter p has to be constrained by $p' = p$. In the following, we denote with $I(V)$ the formula describing the initial configurations, $R(V, V')$ the transition relation, and with $U_i(X)$ the fact that the PTA is in an error state, thus showing unschedulability; we also separate V into the discrete variables D, the continuous clock X_c and X_s state variables, and the parameters P. The PTA.init primitive, given a system of tasks, constructs the I, R, and U formulae. The above formulae are in a fragment of first-order logic, and are interpreted with respect to the background theory of linear constraints over the rationals, with numerical constants and operators having the standard meaning.

Our approach is made practical by the availability of symbolic model checking techniques for infinite state systems. In particular, we rely on NuSMV3, a system for the verification of symbolically described infinite state systems. NuSMV3 is the latest version of NuSMV, build on top of the NuSMV model checker [22] (that is able to deal with finite-state systems using BDD-based and SAT-based verification engines), and the MathSAT [14] solver for Satisfiability Modulo Theories (SMT). For the sake of this work, the capabilities of NuSMV are used "off the shelf".

The reachability problem for a symbolically represented system (including PTA) is undecidable. Thus, in general, the `PTA.reachable` primitive can not be implemented exactly. We thus rely on two kinds of incomplete but complementary sets of engines. The first one is an SMT-generalization of bounded model checking, that is aimed at detecting witness traces of a violation to a given property for a given (bounded) number of steps k. Intuitively, the vector of state variables is replicated k times, thus obtaining $V^0, V^1, \ldots, V^k$, where an assignment to V^i represents the state of the `PTA` after i transitions. In order to find a trace witnessing unschedulability for process i, the transition relation is "unrolled" to obtain the formula

$$I(V^0) \wedge R(V^0, V^1) \ldots \wedge R(V^{k-1}, V^k) \wedge U(V^k)$$

This formula is satisfiable if and only if such a trace exists, and can be fed to MathSAT, that can provide a simulation trace by assigning suitable values to the state variables over time. This is the way the `PTA.reachable` can return a true value (for a given k), and, based on the model found by MathSAT, how `PTA.get_trace` constructs the corresponding trace.

Obviously, the bounded model checking approach can not prove the absence of violations in the general case, and a dual algorithms to conclude the absence of a violation is needed. To this end, NuSMT implements two engines. The first one generalizes k-induction [50]. The other one is an implementation of a CounterExample Guided Abstraction-Refinement (CEGAR) loop, with predicate abstraction computed as in [19], and refinement provided in weakest precondition and SMT interpolation [23]. Both engines are correct (i.e., if they return true, then the property holds), but not complete (i.e., they may return unknown in cases where the property holds).

In the following, we now concentrate on the novel parts, i.e., the `PTA.get_parameter` primitive, that extracts the constraints associated to a trace of length k. Let μ be the assignment to the variables $P, D^0, X^0, D^1, X^1, \ldots, D^k, X^k$ (every parameter retains its value over time). We then simplify the above formula by replacing each discrete variable in $D^0, D^1, \ldots, D^k$ with the corresponding truth value assigned by μ. The result is a formula in the $P, V^0, \ldots, V^k$ variables, that can be seen as symbolic representation of a set of assignments to these variables, each of which represents a different trace, with the same discrete part. In order to find the infeasibility region for the parameters, it is now sufficient to project away the continuous variables by computing the following quantification:

$$\exists X^0, \ldots, X^k . \Phi(P, X^0, \ldots, X^k)$$

In the general case, a general quantifier elimination procedure (such as Fourier-Motzskin elimination) can be required. Several optimizations are possible. First, there exists a functional dependency of continuous variable x^i and the values of the $P, D^0, D^1, \delta^0, \ldots, \delta^{i-1}, D^i$: thus all variables except the δ^i can be eliminated by inlining. The resulting Φ can be decomposed into a conjunction of Φ^i, each depending on $P, \delta^1, \ldots, \delta^i$, so that the computation of the quantification can be reduced to the more structured

$$\Phi^0(P) \wedge \exists \delta^1 . (\Phi^1(P, \delta^1) \wedge \exists \delta^2 . (\ldots \wedge \exists \delta^k . \Phi^k(P, \delta^1, \ldots \delta^k)))$$

In the specific case of periodic activation, we notice that each discrete path is associated with a unique assignment to the δ^i variables, since guards are all in the form of equalities. Thus, the concrete assignment found by MathSAT turns out to be the only possible instantiation for the quantifiers in the above formula, and the constraints over the parameters are simply obtained by value propagation.

Chapter 5

Quinq Tool

We implemented the Parametric Verification of Temporal Properties (PTVP) algorithm for parametric timed automata presented in the previous chapter in the tool named Quinq. In this chapter we first discuss the overview of the tool architecture of Quinq. We then offer the detailed discussion on each of its components: from the specification languages used for the input of Quinq, to the search trace implementation to the parametric analysis algorithm implementation details. We will also offer with some examples on the execution of the algorithm that could help understanding of its details. In the next chapter, we will report on the application of Quinq in different scenarios.

5.1 Design of the tool - an overview

5.1.1 General Architecture

The architecture of the tool Quinq is depicted in Figure 5.1. The main functionalities carried out by our tool are the following:

Input handling In the topmost part of the figure, we show the three possible input possibilities we provide for Quinq. The first one, is by an ordinary NuSMV file that describes the network of PTA as discussed in the previous chapters. For real–time systems consisting of periodic, we have devised a simplified input format, which is translated into NuSMV by an appropriate component (the Automatic periodic Analysis tool). Finally, it is possible to use as an input a UPPAL xml file, which is decorated with additional information (i.e., parameters) and translated into NuSMV by a tool called JUNT.

PTVP algorithm implementation Referring to the PTVP algorithm in Figure 4.1, we have two main parts that need to be implemented:

1. Trace search (the `P`TA.get_trace() function)
2. Sensitivity analysis (the `P`TA.get_parameter() function)

Completion check An inductive reasoning function is provided in NuSMV3. For the purposes of assessing the completion of the algorithm, NuSMV driver can directly interact with NuSMV3 to test whether the currently acquired unschedulability region are maximum.

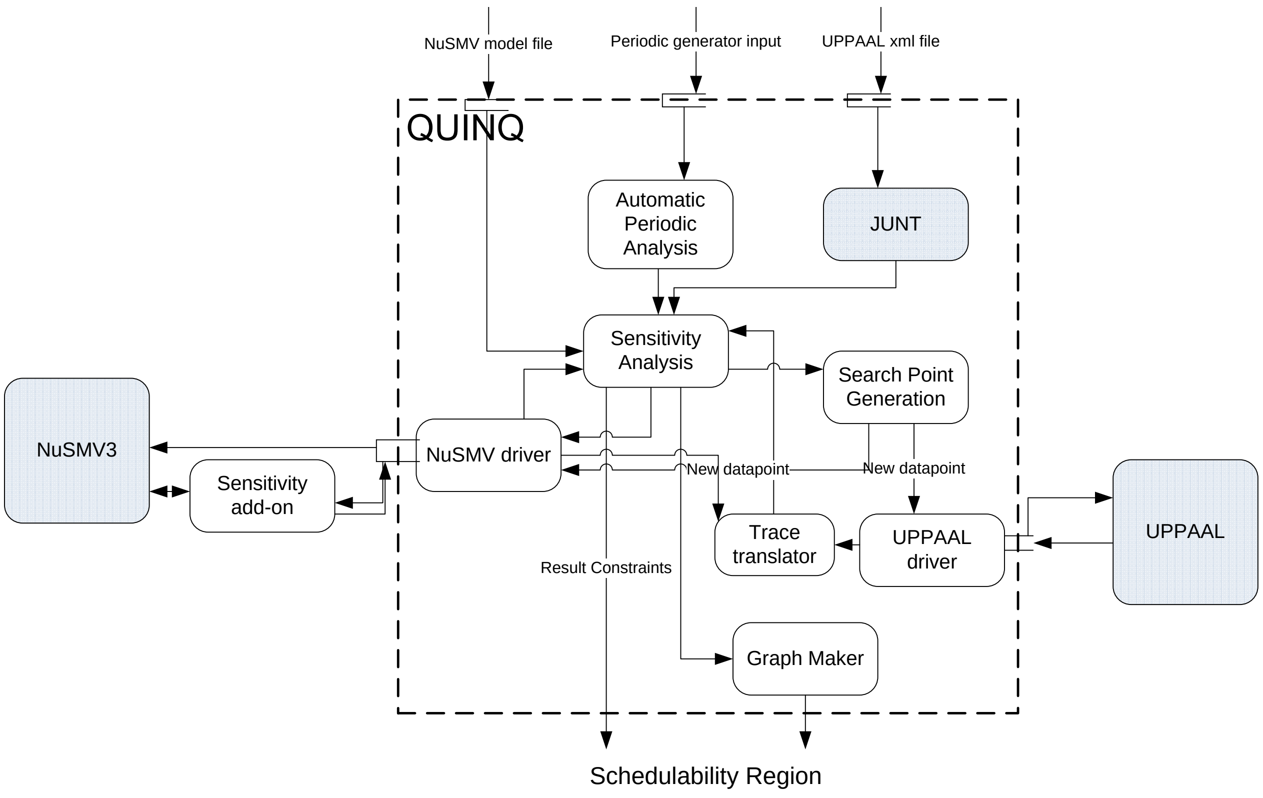


Figure 5.1: General architecture of Quinq

Output handling In the end, after obtaining the result constraints, the schedulability region will be presented in the form of set of constraints in the parameter space directly to the user. Or graph maker component can be called to illustrate the s̈licesö̈f feasibility region in two of the parameters space.

5.1.2 Components

In this section we offer an overview of each component in the architecture of Quinq. The mapping of each component to the functionalities we described above will be given in Table 5.1.2 in the end of this section.

The components can be differentiated in two categories: 1) components used as blackbox, and 2) components we developed. In the following we discuss each components in Quinq architecture that we developed for this work:

Sensitivity Add-on This is the component in which we implemented the main schedulability analysis procedures. As it works in close collaboration with NuSMV3 and their symbolic model checking method, this component is developed as an add-on for NuSMV3. This component provides the following main functions:

1. Sensitivity analysis: Given a model of parametric real-time system and parameters, sensitivity analysis process according to the PTVP algorithm is carried out iteratively to find the schedulability region. BMC search is performed to find the traces, and the sensitivity analysis is carried out on the resulting symbolic trace.
2. Sensitivity analysis from a trace: Provided a model of parametric real-time system, a set of parameters, and a trace from the initial state to one of location representing the temporal violation, this function performs sensitivity analysis to the model based on the trace given. The trace will be replicated in the symbolic model, then the sensitivity analysis will be performed to obtain one unschedulability region that is projected from the trace.

High Level Periodic System Analysis (HLPSA) In this component, the handling for the schedulability analysis is handled completely. This component provides the following functions :

1. Automated input system

   This component handles the automated input system, rather than having the user design their periodic system, he/she only needs to specify the number of tasks involved, the values specification, and the parameters that they desired.
2. Input file generation: parametric and non-parametric

   Based on the input from the user, this component automatically generates the NuSMV input model as well as the corresponding non-parametric model to optimize the search process.
3. Schedulability analysis complete with the search scheme.

   This component then perform the schedulability analysis, in collaboration with the sensitivity addons, as well as the search optimization, in collaboration with the search optimization component.

Search Optimization This component implement the search optimization procedure for obtaining an error trace. Trace finding can be carried out by parametrically using symbolic model checker, or non-parametrically using non-parametric model checkers. Using non-parametric model checker entailed the need to supply fixed points as the parameter instantiation to be checked for a counterexample, i.e. a trace.

This component performs search finding using non-parametric model checker to optimize the trace finding process. It performs the following functions :

1. Point generation

   For the purpose of the search optimization we develop the component to perform the search for instantiation points in the parameter space that potentially lead us to an error trace. We provide three approaches in finding the potential instantiation point :

   - Randomized point

     Provided a limited search area, we can start a random point generator which will return us with points inside the search area with uniform distribution. The intuition is that if no prior information is available on the points that could lead to an error trace, trying random points that are uniformly distributed improves the chance of hitting the area where the error happens.

   - Datagrid

     Provided a limited search area, we can start a search for all the points in the data grid with the granularity that we decide. The maximum granularity of the data search depends on the non-parametric model checker that we use. On the numeric model checker that allows rational number search with the granularity as small as possible is possible. The search can be started from the point with higher values first, or point with lower values first.

   After each point generation a test is carried out to check whether the generated point is already covered in the currently found unfeasibility region.

2. Non-parametric model generation

   Next, this component generates the input file with the parameter variables assigned to the chosen point instantiation. The non-parametric model input file is then provided to the numeric model checker that we choose. As the model generation procedure is closely related to which model checker that we use, this function is directly connected to the numeric model checker driver.

3. Trace translation

   Upon receiving the output from the non-parametric model checker driver then we observe the result. On the case that no error trace is found, the component is then search for the next instantiation point. When an error trace is provided, then a trace translation from the trace into an XML trace readable by NuSMV trace reader.

   During the translation some abstraction process is also performed to the trace as to provide enough information to the NuSMV to replicate the trace, but not too much as to make the trace is highly suited for one and only the instantiation point. Some of the information that must be abstracted are for example the exact time values for the clocks.

Model checker driver This is the component where the direct interaction with the model checkers as blackbox are performed. The function calls to each of the model checker are performed here, as well as the input model generation for the instantiation point we chose. The interpreter of the model checker result is also performed.

Graph generator We provide a component to visualize the result constraints in 2D. Given a set of constraints as the result of Quinq, two of the parameters that we would like to analyse, p_1 and p_2, and the resolution of the graphical result, the tool will provide us with the graph in the space of p_1 and p_2.

In the Figure 5.1 of Quinq architecture several components are shaded to signify the components that have not been developed for this dissertation. The components that we have been using as black box in collaboration with our tool are the following:

NuSMV3 This is the symbolic model checker tool that we employ. It serves several main functions:

- Bounded Model Checking : Perform bounded model checking to symbolic model up to the bound k. Return a counterexample as a witness if the checked property can be falsified in any of the bound from 0 to k.
- Trace replication: Given a trace guide, this function will run the same trace on the symbolic model given.
- Existential elimination : Perform existential elimination of all the variables other than the parameters to a set of constraints, resulting in constraints only in the space of parameters. This function is enabled trough existelim module developed by Mathsat5 developer.
- Inductive Reasoning : This function performs inductive reasoning to the model and the acquired constraints. This function enables us to confirm whether or not we have obtained the complete feasibility region of the system on step-k.

UPPAAL UPPAAL is used mainly to help with the trace finding process. This is done by checking an instantiation of the parametric timed automata (i.e. grounding the parameters to specific values) for a counterexample.

UPPAAL to NuSMV translator (JUNT) This component is developed by Alena Simalatsar for the purpose of COMBEST project [1]. It is a translation path from the Uppaal explicit state model into its symbolic representation in NuSMV, called Java Uppaal to NuSMV Translator (JUNT).

The mapping between each different components to Quinq functionalities are given in the following table:

5.2 Specification Language

We decide to use SMV as our main language,however, we also provide two alternative input possibilities:

1. High level model input: For modelling standard periodic tasks with preemptive, fixed priority scheduler.
2. UPPAAL XML model: We provide a translator for model constructed in UPPAAL to be translated to SMV model up to some extent.

[1](http://www.combest.eu/home/)

Component	*Input handling*	*Main algorithm*	*Completion check*	*Output handling*
`Sensitivity add-on`	–	✓	–	✓
`HLPSA`	✓	✓	✓	–
`Search optimization`	–	✓	–	–
`Model checker driver`	–	✓	✓	–
`Graph generator`	–	–	–	✓
`NuSMV3`	✓	✓	✓	–
`UPPAAL`	–	✓	–	–
`JUNT`	✓	–	–	–

Table 5.1: Mapping between components and functionalities in Quinq

5.2.1 SMV Language

We use SMV as our main specification language for the parametric timed automata input model. As shown in the previous chapter:

1. A parametric real–time system can conveniently be modelled as parametric timed automata, and
2. The SMV language can be used to express this model.

Examples of this modelling technique have been shown in section 3.2.2: Timed Automata Verification in NuSMV.

Clocks and all the continuous variables are represented through real variables. Variables with discrete behaviour can be represented using enumeration. While the rest of the timed automata are represented through boolean variables. Parameters are also represented as variables with the following constraints :

- the variables that we intend to analyse must not be assigned with any values.
- the upper and lower limit to the possible values of the parameter must be given

Further details on the use of the NuSMV input language can be found on its website [2].

5.2.2 UPPAAL language

UPPAAl language can not be directly used in our schedulability analysis framework of parametric timed automata because UPPAAl does not have parameters. However, UPPAAL is a well known modelling language, and it is supported by various tools, including one GUI for modelling. Therefore, we provide the possibility of the flow depicted in Figure 5.2 to use UPPAAL model as an input.

Provided a representation of real–time system expressed as timed automata in UPPAAL, the UPPAAL model can be used as the input to the system. The translator we provide will translate the arbitrary

[2](http://nusmv.fbk.eu/)

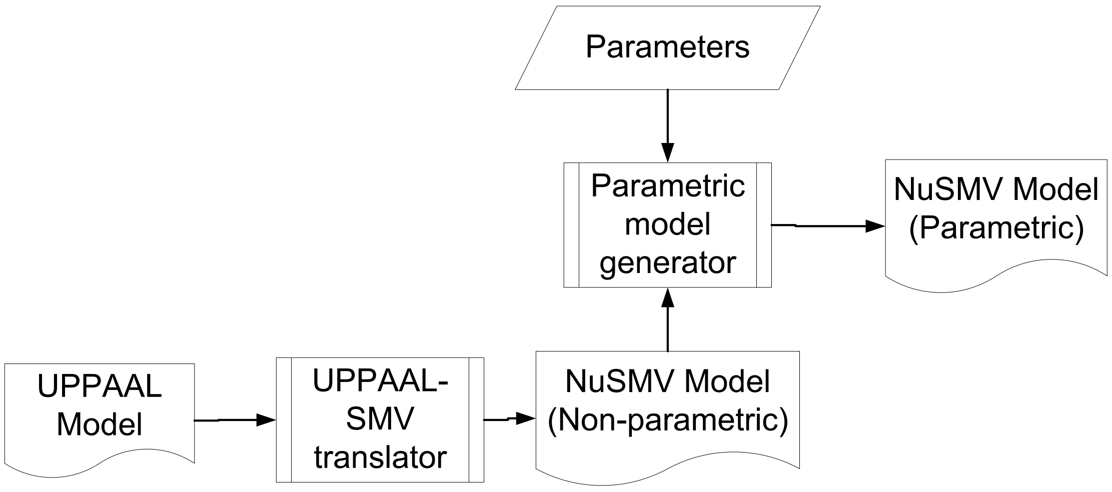


Figure 5.2: The flow of designing input model using UPPAAL model.

UPPAAL model to non-parametric timed automata presented in NuSMV model. With the addition of parameter data by the user, the non-parametric timed automata model is then transformed into subsequent parametric timed automata model in NuSMV that will be the input for our tool.

Enabling the UPPAAL XML file as input provides benefits in twofold: it makes the modelling easier with visual on the model and the resulting UPPAAL model can be used later for the optimization that we will discuss in section **??**. However, some care has to be taken in using this approach as there are several discrepancies between UPPAAL and NuSMV. We discuss the UPPAAL - SMV language discrepancies in the following discussion section. It has to be understood that the main intention of the modelling is to model parametric timed automata in NuSMV and that the UPPAAL model should be used as a guide to at least emulate the behaviour of the desired NuSMV model.

5.2.3 High level modelling of periodic system

Instead of going through the modelling step for the periodic system, we provide a high level view of the periodic system for the user to model . Using periodic plugin, our tool accepts the details such as :

- The number of tasks
- The values for tasks variables : periods, deadline, computation time, offset.
- The variables that are going to be analysed : parameters.
- The upper and lower value limit for the parameters.

This input will then be translated into a template XML file with tags specifying the edges, locations, parameters and non-parameter variable values of the parametric timed automata describing the periodic system and scheduler. This XML file is almost the same as UPPAAL XML file input to describe the timed automata excepts that it has -1 values for variables which are parameters and that there are entries for the lower and upper bound of the parameter values. Thus, later this XML file can also be opened in UPPAAL application. Quinq will then further translate this XML file into NuSMV and perform the requested schedulability analysis on parameters.

5.2.4 Input language discussion

We choose our specification languages by considering the challenges that we presented in section 2.4: the expression power, usability, and tractability of the model. We consider that the expression power of NuSMV language are sufficient to represent parametric timed automata in general, and subsequently real–time system expressed through timed automata. It enables us to represent all possible desired properties, as well as unlimited possibilities in synchronization mechanism between timed automata.

Modelling parametric timed automata for the first times in NuSMV requires a basic background knowledge on modelling formal system. For that purpose, we enable the high level modelling for standard periodic cases. Aside from increasing its usability it also increases the tractability of the model. Using automatic model generation we provided reducing the modelling effort into just specifying the number of tasks and their design variables values. This applies for all systems from small number of tasks and applicable too for system with significant number tasks.

On arbitrary model of timed automata, where more complex activation pattern and scheduler model is needed, we also provide a simple input alternative by enabling UPPAAL XML model as an input guide for the NuSMV model.Starting the design in the UPPAAL will enable the user to have a flavour of the system correctness on non-parametric timed automata domain. Translating the UPPAAL XML model to NuSMV will then provide user with the NuSMV model with the same behaviour that can be used as the basic to further improvise according to the desired parametric behaviour. This alternative increases the usability for Quinq users that are the first time user of NuSMV.

However there are some discrepancies on SMV and UPPAAL semantics that are relevant for our modelling domain :

1. Integer vs Real domain

   In NuSMV language we work using the rational domain. This is particularly suitable for the domain of parametric real–time system as we could express the continuous notion of time seamlessly. In contrary, all the modelling in UPPAAL is done in the integer domain. Values for every variables are thus expressed in integer. This makes the modelled system to have discrete behaviour as all the possible variables valuation are in integer. Thus, some abstraction is needed in modelling parametric real time system in UPPAAL as to fit the integer behaviour.

2. Array data structure

   Arrays in UPPAAL are provided, this then will be translated to variables in NuSMV, however the operation to the array as a whole needs to be ensured that it has been properly translated. Arrays are particularly challenging to translate, because array support in NuSMV is very limited. Some work around might be needed to ensure the correct behaviour has been implemented. For example, when arrays are used to represent the transition guards or assignments in Uppaal, we need to introduce a number of transitions equal to the array size in the NuSMV module. There might be a case when array size is defined by a template parameter that can vary from one template instantiation to another. This causes the translation of one Uppaal transition into an unfixed number of NuSMV transitions. Therefore, different instantiations of the same template should be translated into different NuSMV modules which cause the NuSMV code to expand.

   This lack of data structure entails some work around into implementing routines involving buffer and array of similar items. With this the queue behaviour for example must be implemented by providing certain macros to emulate the behaviour of push and pop in a queue / stack.

3. Clocks

As we have explained in chapter 4, the clocks that represent time are modelled using real variables. Thus the expected behaviour of a clock that only get increased by values unless reset or assigned new values, and never takes value below zero has to be ensured by constraints to the variables. Incomplete constraints specification for the clock variable might cause the assignment of the clocks at each steps to vary arbitrarily.

The clocks are modelled automatically in UPPAAL as a type clock. These clocks will increase its value on its own as time elapses, and could be reset as per guard instruction. However, as it is strictly formulated for clock purposes, the only operations that are allowed are comparison with another variable / value and reset operation. The clock values can not be obtained to be further operated in a linear operation with other variables for example. As such on timed automata where we need the clock values to be operated on, some counters must be established to keep the clock values.

4. Transition synchronization

   Since the transitions are essentially presented as a variable that could take values within certain possible transitions, we need to properly specify which value can be taken at what time. Synchronization between transitions that may or may not be asynchronized would need some if-else case to model.

   The synchronization in UPPAAL happens through channels. The automata synchronize via the channel by sending and receiving information in the channel. The channel in some ways can be seen as label representation in classical timed automata. It needs to be ensured that the synchronization has been represented correctly both in UPPAAL and NuSMV version. The channel encoding in UPPPAAL in several ways limit the synchronization possibility as opposite to the all possible synchronization in NuSMV.

All the aspects above have to be taken care of when the flow depicted in Figure 5.2 for input using UPPAAL model is used. In the long run, the modelling using NuSMV is more advisable for our parametric schedulability analysis framework. It would provide the most freedom in modelling the system and represent the authentic continuous behaviour of the real–time system.

5.3 PTVP Algorithm Implementation

We built the tool Quinq based on the iterative PTVP algorithm presented in Figure 4.1. The tool Quinq will need the parametric timed automata model of the system in SMV file as an input. This input file describes the real–time system through a parametric timed automata network, describing the activation of tasks in the system and the scheduler for the tasks. At least one error location should be included to indicate whether the system has violated its deadline or any of its desired temporal behaviour.

Quinq also requires a parameter file input. It is a text file with a list of variables in the smv model input that we would like to treat as parameters. Several different options to run the main functions are also provided in the tool. Such as the option to only run one iteration of the algorithm, or to only perform sensitivity analysis given a specific trace. We offer an overview of the tool functions in Appendix A.

Performing feasibility analysis of one task can be focused on working on one smv input file as we can see in Figure 5.3. The additional input to the model file that we need are:

1. BMC length : up to what length would we like to perform the BMC for error traces.

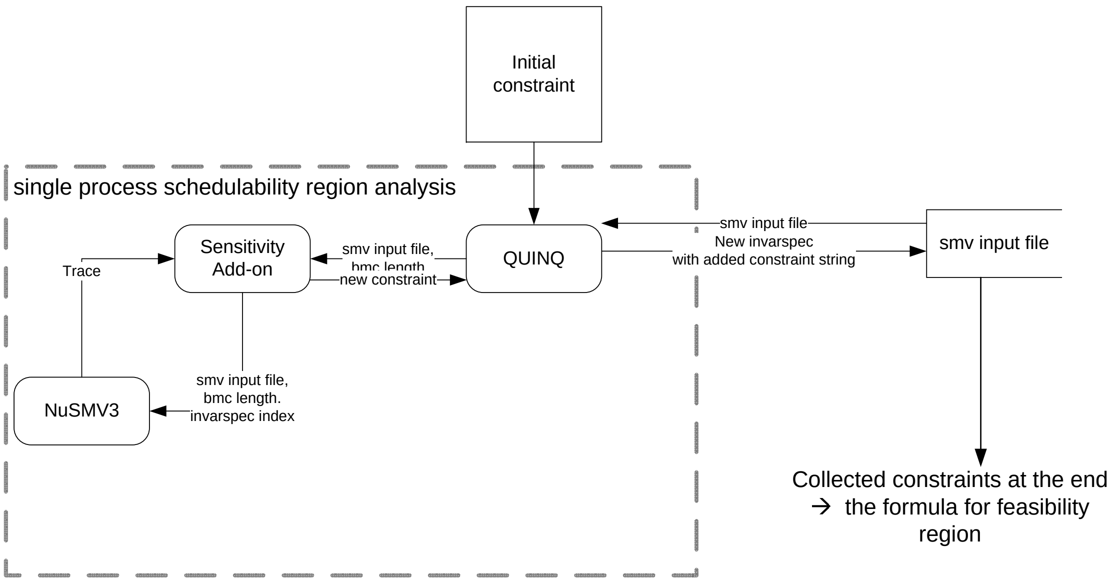

Initial constraint
single process schedulability region analysis
Sensitivity Add-on
smv input file, bmc length
new constraint
QUINQ
smv input file
New invarspec with added constraint string
smv input file
Trace
NuSMV3
smv input file, bmc length. invarspec index
Collected constraints at the end → the formula for feasibility region

Figure 5.3: Component interactions in performing feasibility analysis implementation for one file model.

2. Initial constraints : these are the constraints on the search space of the parameters. On the first iteration this is the initial upper and lower bound of the parameter. On the next iterations this is the conjunction between the initial upper and lower boundary and the constraints found in the previous iteration.

3. Property to be checked : this is included in the model. For our case of schedulability this is the reachability property of the bad or error states where the system will violate the tasks deadline or any of the system requirements.

For one sensitivity analysis process of each smv file, Quinq evokes the sensitivity addons in NuSMV3 that is the driver for NuSMV3 in performing the implementation for the algorithm. The processes performed by the sensitivity add-on is the following:

- Error trace search
- Sensitivity analysis

The illustration of the two main phases above can be seen in Figure 5.4.

5.3.1 Error trace search

In the error trace search, we provide two alternatives of performing the search :

Parametric search In here Quinq invokes NuSMV functions to read the model file input and perform the BMC search. Specifically we invoke NuSMV incremental BMC function to find a trace on the parametric automata unrolled incrementally from length 0 (initial state) up to k-length, where k is the BMC length given in the input. We perform verification on satisfiability of the reachability

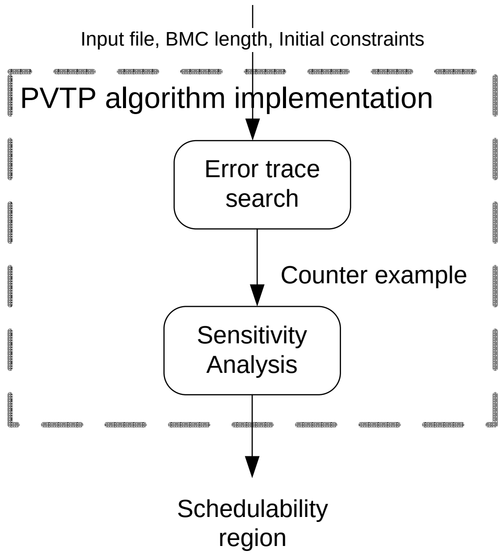


Figure 5.4: A flow for the feasibility analysis implementation for one process in sensitivity add-on.

property on the unrolled model, i.e. a set of value assignment to all the variables which describes a path where a deadline will be violated and the task system is not schedulable. In every iteration the model is built observing the initial constraints given. Each time this initial constraints are conjuncted with the new constraints result from the previous iteration.

On the case of finding a counter example for the SAT, an error path, the result trace, is dumped in a NuSMV trace file in XML format as well as transferred through internal data structure to the next process. On the case that the problem is SAT then we obtain the result declaring that the model will not reach any error state up to k-step. Using BMC for our verification we need to do this verification for each increasing bound until we reach SAT for step k.

Example 5.1. *We use the system illustrated in Figure 3.5 encoded in NuSMV as parametric timed automata. Using the following assignment* $\{T1 = D1 = 4, T2 = D2 = 6, offset1 = offset2 = 0\}$*, and the computation time* $C1$ *and* $C2$ *as the parameters. For the simplicity of the result we run here the check for the schedulability of task* τ_1*. Given a model consisting of task* τ_1 *and scheduler for* τ_1*. We provide no initial constraint, i.e.* ***True*** *is the initial constraint. A counter example is then found. A trace from the NuSMV BMC function is the following :*

```
Trace Description: MSAT BMC Counterexample
Trace Type: Counterexample
  -> State: -1.1 <-
    active_1 = TRUE
    D1 = 4
    T1 = 4
    C1 = 7
    offset1 = 0
    scheduler.C1 = 7
    scheduler.D = 4
    process_1.period = 4
    process_1.offset = 0
    scheduler.state = idle
    scheduler.transition = idle_check
    scheduler.clk = 0
    scheduler.r = 0
    process_1.state = wait_for_offset
    process_1.clk = 0
    process_1.transition = offset_release
    the_time = 0
    delta_t = 0
  -> State: -1.2 <-
    scheduler.state = check
    scheduler.transition = time_elapse
    scheduler.r = 7
    process_1.state = wait_for_period
    process_1.transition = time_elapse
    delta_t = 4
  -> State: -1.3 <-
    scheduler.transition = check_error
    scheduler.clk = 4
    process_1.clk = 4
    process_1.transition = stutter
    the_time = 4
    delta_t = 0
  -> State: -1.4 <-
    scheduler.state = error
    scheduler.transition = time_elapse
    process_1.transition = time_elapse
    delta_t = 1
```

*The trace in the state 0 shows the initial state of the timed automata. The trace assigned $C1$ with the value 7. The transition that is taken first is offset release: a job τ_1 is activated. In the next state a time elapse transition with $delta\_t = 4$ is taken, increasing the clocks values to 4. At state 3, a transition taking the system to error state is launched because the deadline is reached $clk = D1$ yet the job τ_1 with computation time $C1 = 7$ has not finished its execution. Thus, in the last state, we see that the system is now in the error location. This is the counter example for our problem, that with the initial constraint **True** for $C1$ the system will be able to reach the error state in 4 states, or 3 steps.*

Non-parametric model search The PTVP algorithm implementation that relies on the symbolic model checker through BMC to enumerate all the unschedulable cases works effectively to find each possible error trace in each BMC step. However, using only symbolic model checker to solve the problem take some considerable time as it needs to prove UNSAT for every step of the bounded model checking it performs. The checking of a single numeric point in the search space of the system in the other hands is very fast and simple. And later the trace still can be generalized by our tool, using symbolic model checker, to result in a part of the schedulability region. Theoretically, we could get the whole schedulability region of the system if we could enumerate all the points in the search space, feed each of these points to a checker and analyse the error trace for each point that is unschedulable.

The idea of encompassing non-parametric timed automata model checker can be seen in Figure 5.5. We have two flows going in the tool :

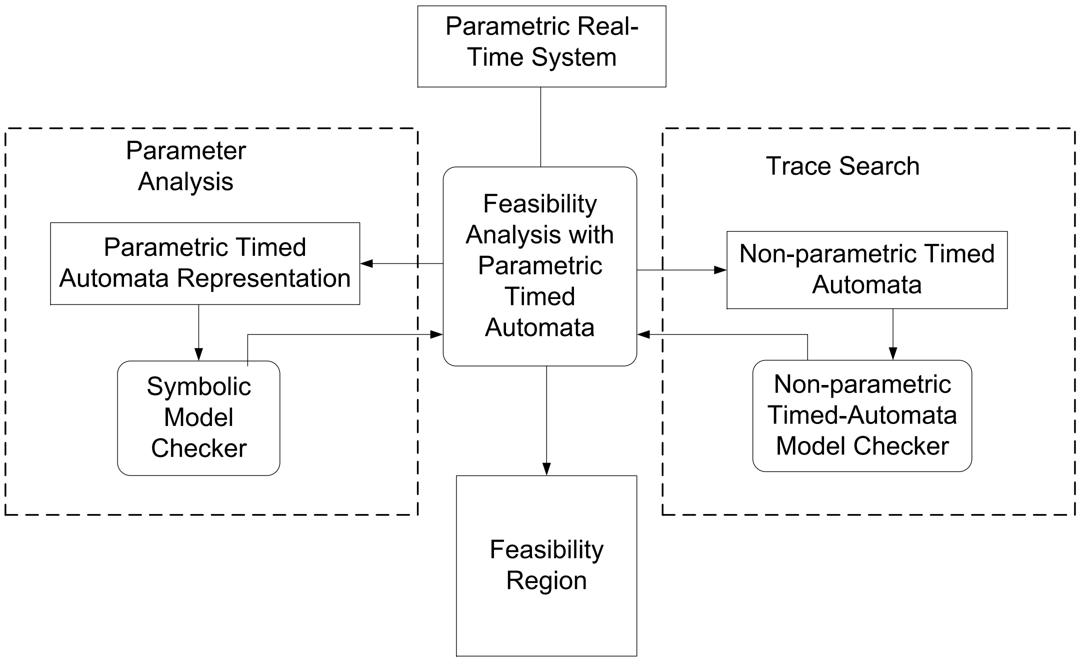

Parametric Real-Time System
Parameter Analysis
Parametric Timed Automata Representation
Symbolic Model Checker
Feasibility Analysis with Parametric Timed Automata
Trace Search
Non-parametric Timed Automata
Non-parametric Timed-Automata Model Checker
Feasibility Region

Figure 5.5: A template of implementation using parametric and non-parametric model checker as black box

1. Search optimization flow
   In the search optimization flow we substitute the previous approach of using BMC on the phase of error trace search with non-parametric model checker. Instead of BMC search we added a procedure to find an instantiation point which go to the error state and return us with an error trace.
2. Parameter analysis flow
   In the parameter analysis flow we employ the symbolic model checker to first replicate the trace given by the search optimization flow in the symbolic model, then continue the remaining phases of parameter analysis on the trace.

However, this, too, is an inefficient way of solving the problem. First, there are too many points in the search space to enumerate, and second, as a trace would result in a region of unschedulability which covers a substantial number of point, it would be the case that checking and analysing a point already covers the region of other points in its proximity.

We devise instead several approaches to bridge these two approaches. We divide the search space into data grids with certain granularity that we can set. We then enumerate points within these grids and then feed any of these to the non-parametric timed automata model checker only if it is not yet contained in the obtained schedulability region so far. Doing this in iteration provides some initial constraints that limit the search space into lesser search space for schedulability region. We can then further continue our analysis using symbolic model checker in the remaining area.

The complete flow with details on the exchanged data between UPPAAL and NuSMV when we use UPPAAL as non-parametric model checker to perform the search is illustrated in figure 5.6.

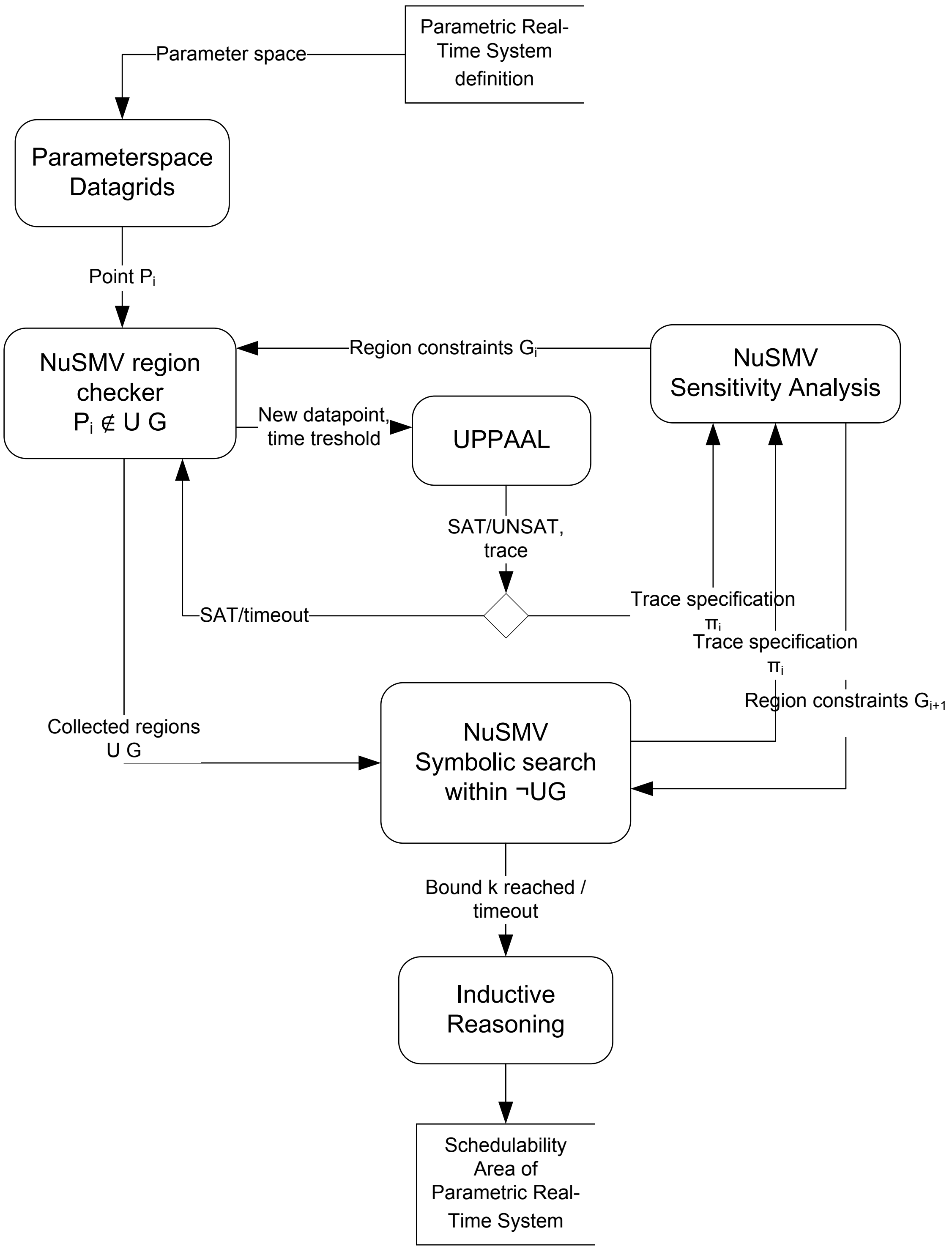

Parametric Real-Time System definition
Parameter space
Parameterspace Datagrids
Point P_i
NuSMV region checker $P_i \notin \cup G$
Region constraints G_i
NuSMV Sensitivity Analysis
New datapoint, time treshold
UPPAAL
SAT/UNSAT, trace
SAT/timeout
Trace specification π_i
Trace specification π_i
Region constraints G_{i+1}
Collected regions $\cup G$
NuSMV Symbolic search within $\neg \cup G$
Bound k reached / timeout
Inductive Reasoning
Schedulability Area of Parametric Real-Time System

Figure 5.6: A flow for interaction between UPPAAL and NuSMV for the search optimization scheme.

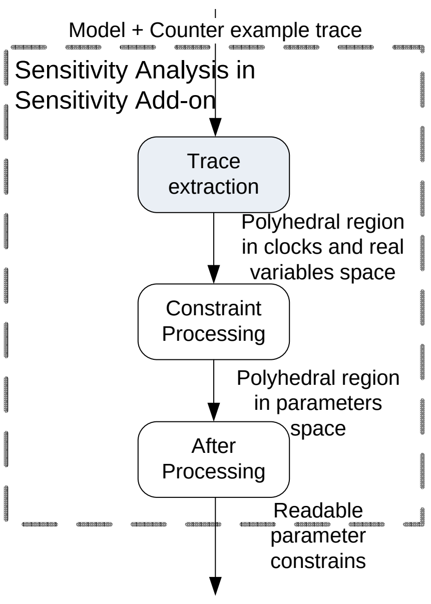

Model + Counter example trace
Sensitivity Analysis in Sensitivity Add-on
Trace extraction
Polyhedral region in clocks and real variables space
Constraint Processing
Polyhedral region in parameters space
After Processing
Readable parameter constrains

Figure 5.7: A flow for parameter projection from the symbolic model and trace in sensitivity analysis phase.

5.3.2 Sensitivity analysis

From the obtained symbolic trace and model, we need to project out the clocks, other variables, and the boolean values so that we are left with a kernel of constraints on the specified parameters. This amounts to an existential quantifier elimination for all variables other than the parameters in the symbolic trace.

The obtained trace is a sequence of assignments and constraints on boolean variables and other variables (clocks, variables and parameters). Forcing the assignment of boolean variables in every steps to a specified value according to the trace amounts to a dictate a set of discrete transitions to be taken in the model. Using the propagation of the boolean variables assignments in the trace we extract a set of constraints in real variables and parameters on our model that correspondences to the trace.

Generally speaking we get a polyhedron $Polyhedron$ defined in a space consisting of the parameters (which we succinctly denote as $\mathbf{P}$) and of the variables $\mathbf{X}$ containing all the clocks in the system, i.e. $Polyhedron\,\{\mathbf{P},\mathbf{X}\}$.

The purpose of this step is to eliminate the dependence off $\mathbf{X}$ variables, i.e. projecting a Polyhedron $Polyhedron'$, where

$$Polyhedron'(\mathbf{P}) \leq 0 \Longleftrightarrow \exists \mathbf{X} | Polyhedron(\mathbf{P},\mathbf{X}) \leq 0$$

Thus, our problem is a quantifier elimination or (from a geometric point of view) a projection of $\mathbf{P}$ into a subspace.

We implement this sensitivity analysis in our tool with the following flow of phases, depicted in Figure 5.7. These process can be done with any quantifier elimination method such as Fourier Motzkin method, as well as the approach that we use and describe below:

Trace extraction From the previous step we obtain a trace from the BMC function, which is the SAT counterexample path from the verification. This contains the values for each boolean and real variables in the task system formulation which leads to a path which is not schedulable. In this step our purpose is to find from these assignments a polyhedron in the space of clocks and real variables involved in the model. This phase is shaded in Figure 5.7 as this component is developed by NuSMV developers. We need to do the following :

1. Unrolling the symbolic model to the k-step

   This is done to provide us with the whole initial view of the system, where all possible valuations for all the booleans and real variables are available.

   Example 5.2. *To provide a flavour of the size of an unrolled model, we present here an example of the unrolling on the step 0, which is the initial state, of the model we worked on in example 5.1. The unrolled model for step 0 is as follows :*

```
(@0{(((delta_t = 0.0 | delta_t > 0.0) & (the_time >= 0.0 & (process_1.clk
    >= 0.0 & ((process_1.offset >= 0.0 & (process_1.state =
    wait_for_offset -> process_1.clk <= process_1.offset)) &
    ((process_1.state = wait_for_period -> process_1.clk <=
    process_1.period) & (scheduler.clk >= 0.0 & ((!(scheduler.transition =
    stutter & process_1.transition = stutter) & ((((scheduler.transition =
    busy_idle | scheduler.transition = check_idle) | scheduler.transition
    = check_error) <-> process_1.transition = stutter) &
    ((((((scheduler.transition = idle_busy_eq | scheduler.transition =
    idle_check) | scheduler.transition = busy_busy_eq) |
    scheduler.transition = busy_check) | scheduler.transition =
    check_check_eq) <-> (process_1.transition = period_release |
    process_1.transition = offset_release)) & ((!all_time_elapse ->
    delta_t = 0.0) & ((all_time_elapse <-> process_1.transition =
    time_elapse) & (all_time_elapse -> delta_t > 0.0)))))) &
    ((scheduler.state = error -> (all_time_elapse & delta_t = 1)) &
    ((process_1.offset = offset1 & offset1 = 0.0) & ((scheduler.C1 = C1 &
    C1 > 0.0) & ((process_1.period = T1 & T1 = 4.00) & ((!active_1 ->
    (process_1.state = inactive & (process_1.transition = stutter |
    process_1.transition = time_elapse))) & (scheduler.D = D1 & D1 =
    4.00)))))))))))) & (((next(the_time) := the_time + delta_t) &
    (((process_1.transition = stutter -> next(process_1.clk) =
    process_1.clk) & ((process_1.transition = period_release ->
    next(process_1.clk) = 0.0) & ((process_1.transition = time_elapse ->
    next(process_1.clk) = process_1.clk + delta_t) & (process_1.transition
    = offset_release -> next(process_1.clk) = 0.0)))) &
    (((process_1.transition = period_release -> next(process_1.state) =
    wait_for_period) & ((process_1.transition = offset_release ->
    next(process_1.state) = wait_for_period) & ((process_1.transition =
    stutter -> next(process_1.state) = process_1.state) &
    (process_1.transition = time_elapse -> next(process_1.state) =
    process_1.state)))) & ((process_1.transition = offset_release ->
    (process_1.state = wait_for_offset & process_1.clk =
    process_1.offset)) & ((process_1.transition = period_release ->
    (process_1.state = wait_for_period & process_1.clk =
    process_1.period)) &
((next(scheduler.r) := case
(scheduler.transition = idle_busy_eq | scheduler.transition = idle_check)
    : scheduler.C1;
scheduler.transition = busy_busy_eq : scheduler.r + scheduler.C1;
scheduler.transition = busy_check : (scheduler.r + scheduler.C1) -
    scheduler.clk;
TRUE : scheduler.r;
```

```
esac) &
((next(scheduler.clk) := case
((scheduler.transition = idle_busy_eq | scheduler.transition =
    idle_check) | scheduler.transition = busy_check) : 0.0;
scheduler.transition = time_elapse : scheduler.clk + delta_t;
scheduler.transition = idle_busy_hil : 0.0;
TRUE : scheduler.clk;
esac) & ((scheduler.transition = time_elapse ->
    next(scheduler.transition) != time_elapse) & (((scheduler.transition =
    check_idle -> (scheduler.state = check
& (scheduler.clk = scheduler.r & scheduler.clk <= scheduler.D))) &
    (scheduler.transition = check_error -> (scheduler.state = check &
    (scheduler.clk < scheduler.r & scheduler.clk = scheduler.D)))) &
    ((scheduler.transition = check_error -> next(scheduler.state) = error)
    & ((scheduler.transition = check_check_eq -> next(scheduler.state) =
    check) & ((scheduler.transition = check_idle -> next(scheduler.state)
    = idle) & ((scheduler.transition = busy_check -> next(scheduler.state)
    = check) & ((scheduler.transition = busy_busy_eq ->
    next(scheduler.state) = busy) & ((scheduler.transition = busy_idle ->
    next(scheduler.state) = idle) & ((scheduler.transition = idle_check ->
    next(scheduler.state) = check) & ((scheduler.transition = idle_busy_eq
    -> next(scheduler.state) = busy) & ((scheduler.transition =
    time_elapse -> next(scheduler.state) = scheduler.state) &
    ((scheduler.transition = stutter -> next(scheduler.state) =
    scheduler.state) & ((scheduler.transition = busy_idle ->
    (scheduler.state = busy & scheduler.clk = scheduler.r)) &
    ((scheduler.transition = check_check_eq -> scheduler.state = check) &
    ((scheduler.transition = busy_check -> scheduler.state = busy) &
    ((scheduler.transition = busy_busy_eq -> scheduler.state = busy) &
    ((scheduler.transition = idle_busy_eq -> scheduler.state = idle) &
    (scheduler.transition = idle_check -> scheduler.state =
    idle)))))))))))))))))))))))))) & next(((delta_t = 0.0 | delta_t > 0.0)
    & (the_time >= 0.0 & (process_1.clk >= 0.0 & ((process_1.offset >= 0.0
    & (process_1.state = wait_for_offset -> process_1.clk <=
    process_1.offset)) & ((process_1.state = wait_for_period ->
    process_1.clk <= process_1.period) & (scheduler.clk >= 0.0 &
    ((!(scheduler.transition = stutter & process_1.transition = stutter) &
    ((((scheduler.transition = busy_idle | scheduler.transition =
    check_idle) | scheduler.transition = check_error) <->
    process_1.transition = stutter) & ((((((scheduler.transition =
    idle_busy_eq | scheduler.transition = idle_check) |
    scheduler.transition = busy_busy_eq) | scheduler.transition =
    busy_check) | scheduler.transition = check_check_eq) <->
    (process_1.transition = period_release | process_1.transition =
    offset_release)) & ((!all_time_elapse -> delta_t = 0.0) &
    ((all_time_elapse <-> process_1.transition = time_elapse) &
    (all_time_elapse -> delta_t > 0.0)))))) & ((scheduler.state = error ->
    (all_time_elapse & delta_t = 1)) & ((process_1.offset = offset1 &
    offset1 = 0.0) & ((scheduler.C1 = C1 & C1 > 0.0) & ((process_1.period
    = T1 & T1 = 4.00) & ((!active_1 -> (process_1.state = inactive &
    (process_1.transition = stutter | process_1.transition =
    time_elapse))) & (scheduler.D = D1 & D1 = 4.00))))))))))))))))} &
    @0{((init(the_time) := 0.0) & (process_1.clk = 0.0 & (process_1.offset
    >= 0 & (process_1.period > 0 & ((init(scheduler.r) := 0.0) &
    ((init(scheduler.clk) := 0.0) & (scheduler.state = idle &
    (scheduler.C1 >= 0.0 & (active_1 & (active_1 -> process_1.state =
     wait_for_offset))))))))))})
```

All the formula making up the above "blob" dictate the constraints for every variables, boolean and real, value assignment at time zero. As this is just the unrolling of step 0, at the unrolling of step 4 we will have set of constraints with the size 4 times as much as the size of this unrolling. This is because we have four copies of all the constraints on every variables, one for each time step.

2. Substituting values according to the trace

   The unrolled model is then s̈licedb̈y the trace that we obtained. We do it by assigning all the boolean and real variables based on the traces. Thus from all possible unrolled model we have ourselves a

   Example 5.3. *Continuing working on example model of Figure 3.5 with counter example presented in example 5.1, some of the assignments by the trace for variables at time 0 are the following :*

```
time 0 : delta_t = 0
time 0 : the_time = 0
time 0 : process_1.clk = 0
time 0 : process_1.state = wait_for_offset
time 0 : scheduler.clk = 0
time 0 : scheduler.transition = idle_check
time 0 : process_1.transition = offset_release
time 0 : scheduler.state = idle
time 0 : scheduler.r = 0
```

3. Propagating boolean values

   Next we propagate the assignments on boolean variables to simplify the model that we have.

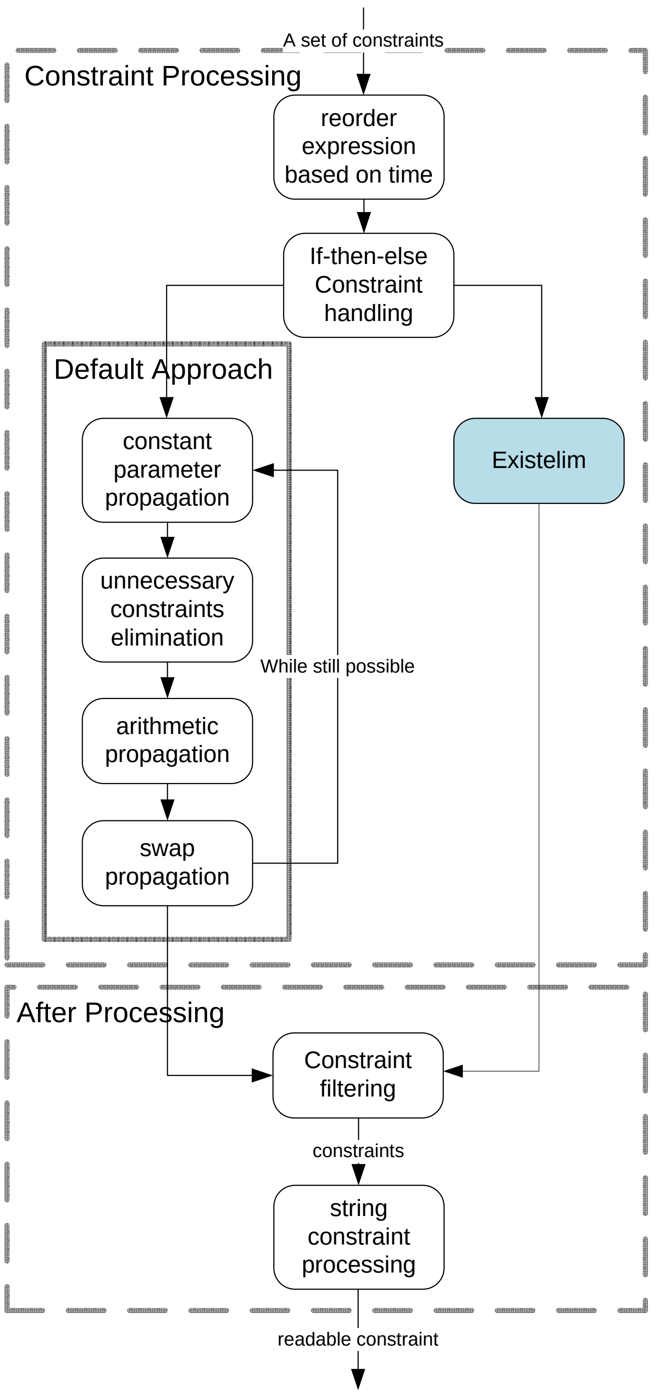
   Example 5.4. *Propagating the boolean assignments given in the trace, such as the assignment for variables at time 0 given in example 5.3, we have for example the following set of constraints as the result:*

```
          ((((@2{delta_t} = 0.0 | @2{delta_t} > 0.0) & (@2{the_time} >= 0.0
              & (@2{process_1.clk} >= 0.0 & (@0{process_1.offset} >= 0.0 &
              (@2{process_1.clk} <= @0{process_1.period} &
              (@2{scheduler.clk} >= 0.0 & (@2{delta_t} = 0.0 &
              ((@0{process_1.offset} = @0{offset1} & @0{offset1} = 0.0) &
              ((@0{scheduler.C1} = @0{C1} & @0{C1} > 0.0) &
              ((@0{process_1.period} = @0{T1} & @0{T1} = 4.00) &
              (@0{active_1} & (@0{scheduler.D} = @0{D1} & @0{D1} =
              4.00))))))))))))) & ((@3{the_time} = @2{the_time} + @2{delta_t}
              & (@3{process_1.clk} = @2{process_1.clk} & (@3{scheduler.r} =
              @2{scheduler.r} & ((@2{scheduler.clk} < @2{scheduler.r} &
              @2{scheduler.clk} = @0{scheduler.D}) &
@3{scheduler.clk} = @2{scheduler.clk})))) & ((@3{delta_t} = 0.0 |
     @3{delta_t} > 0.0) & (@3{the_time} >= 0.0 & (@3{process_1.clk} >= 0.0
     & (@0{process_1.offset} >= 0.0 & (@3{process_1.clk} <=
     @0{process_1.period} & (@3{scheduler.clk} >= 0.0 & (@3{delta_t} > 0.0
     & (@3{delta_t} = 1 & ((@0{process_1.offset} = @0{offset1} &
     @0{offset1} = 0.0) & ((@0{scheduler.C1} = @0{C1} & @0{C1} > 0.0) &
     ((@0{process_1.period} = @0{T1} & @0{T1} = 4.00) & (@0{active_1} &
     (@0{scheduler.D} = @0{D1} & @0{D1} = 4.00)))))))))))))))) &
     ((((@1{delta_t} = 0.0 | @1{delta_t} > 0.0) & (@1{the_time} >= 0.0 &
     (@1{process_1.clk} >= 0.0 & (@0{process_1.offset} >= 0.0 &
     (@1{process_1.clk} <= @0{process_1.period} &
 (@1{scheduler.clk} >= 0.0 & (@1{delta_t} > 0.0 & ((@0{process_1.offset} =
     @0{offset1} & @0{offset1} = 0.0) & ((@0{scheduler.C1} = @0{C1} &
     @0{C1} > 0.0) & ((@0{process_1.period} = @0{T1} & @0{T1} = 4.00) &
     (@0{active_1} & (@0{scheduler.D} = @0{D1} & @0{D1} = 4.00))))))))))))))
     & (((@2{delta_t} = 0.0 | @2{delta_t} > 0.0) & (@2{the_time} >= 0.0 &
     (@2{process_1.clk} >= 0.0 & (@0{process_1.offset} >= 0.0 &
     (@2{process_1.clk} <= @0{process_1.period} & (@2{scheduler.clk} >= 0.0
     & (@2{delta_t} = 0.0 & ((@0{process_1.offset} = @0{offset1} &
     @0{offset1} = 0.0) & ((@0{scheduler.C1} = @0{C1} & @0{C1} > 0.0) &
     ((@0{process_1.period} = @0{T1} & @0{T1} = 4.00) & (@0{active_1} &
     (@0{scheduler.D} = @0{D1} & @0{D1} = 4.00)))))))))))))) & (@2{the_time}
     = @1{the_time} + @1{delta_t} & (@2{process_1.clk} = @1{process_1.clk}
     + @1{delta_t} & (@2{scheduler.r} = @1{scheduler.r} & @2{scheduler.clk}
     = @1{scheduler.clk} + @1{delta_t}))))) & ((((@0{delta_t} = 0.0 |
     @0{delta_t} > 0.0) & (@0{the_time} >= 0.0 & (@0{process_1.clk} >= 0.0
     & ((@0{process_1.offset} >= 0.0 & @0{process_1.clk} <=
     @0{process_1.offset}) & (@0{scheduler.clk} >= 0.0 & (@0{delta_t} = 0.0
     & ((@0{process_1.offset} = @0{offset1} & @0{offset1} = 0.0) &
     ((@0{scheduler.C1} = @0{C1} & @0{C1} > 0.0) & ((@0{process_1.period} =
     @0{T1} & @0{T1} = 4.00) & (@0{active_1} & (@0{scheduler.D} = @0{D1} &
     @0{D1} = 4.00))))))))))) & (((@1{delta_t} = 0.0 | @1{delta_t} > 0.0) &
     (@1{the_time} >= 0.0 & (@1{process_1.clk} >= 0.0 &
     (@0{process_1.offset} >= 0.0 & (@1{process_1.clk} <=
     @0{process_1.period} & (@1{scheduler.clk} >= 0.0 & (@1{delta_t} > 0.0
     & ((@0{process_1.offset} = @0{offset1} & @0{offset1} = 0.0) &
     ((@0{scheduler.C1} = @0{C1} & @0{C1} > 0.0) & ((@0{process_1.period} =
     @0{T1} & @0{T1} = 4.00) & (@0{active_1} & (@0{scheduler.D} = @0{D1} &
     @0{D1} = 4.00)))))))))))) & (@1{the_time} = @0{the_time} + @0{delta_t}
     & (@1{process_1.clk} = 0.0 & (@0{process_1.clk} = @0{process_1.offset}
     & (@1{scheduler.r} = @0{scheduler.C1} & @1{scheduler.clk} = 0.0))))))
     & (@0{the_time} = 0.0 & (@0{process_1.clk} = 0.0 &
     (@0{process_1.offset} >= 0 & (@0{process_1.period} > 0 &
     (@0{scheduler.r} = 0.0 & (@0{scheduler.clk} = 0.0 & (@0{active_1} &
     @0{scheduler.C1} >= 0.0))))))))))
```

As we can see, the size of this set of constraints has been greatly reduced, and we no longer see the constraints for the locations for example, as the location has been assigned according to the trace. However, we still have rather large set of constraints at hand, and the set still contains several variables that are not parameters.

The end result of this phase is a model of the trace in the system model expressed with clocks and real variables.

The processing of the symbolic constraints are explained in Figure 5.8. In this phase from the model we obtained previous step that intersects the system model and the counterexample we



Constraint processing

Figure 5.8: Phases in constrains processing and after processing.

extract a smaller model containing only relevant real variables: the parameters that we would like to analyse, and some constants.

It started with reordering the constraints clauses based on time order. This is necessary to be done so the inference between constraints can be started top-down ordered by their timestamp. The second step is to handle the if-then-else clauses (ITE clauses) to simplify all the constraints into conjunction of inequality or equality clauses. This is done by evaluating all the ITE clauses. First we collect all of the guards containing the ITE clauses with G_i as the condition of the ITE clause. We then plug this G_i to the model and perform the satisifiability test with G_i as the property. Next, we perform the same satisfiability test to the property $\neg G_i$. Evaluating the condition clause G_i is then done by assigning **True** to G_i if the G_i property is satisfiable and $\neg G_i$ is not satisfiable and vice versa. If it is found that both G_i and $\neg G_i$ are both satisfiable or both not satisfiable, then Quinq will report that the guard G_i is inconclusive or that there is an error in the model respectively. The result of this function will leave us with a set of clauses connected with conjunctions.

For the next constraint processing we have two approaches :

- Default approach

  The default approach employs the encoding of the following processes as expressed in Figure 5.8. This process consists of:

  1. Constant parameter propagation

     Collecting all the clauses with equality signs, we propagate the assignment of variables to constant or parameter or a set of operation to constant and parameters that we would like to analyse. Iteratively this will omit all variables in the set of clauses other than the parameters and constants.

     Example 5.5. *Some of the variables assignment that we obtained in the first step of constant parameter propagation for the model in example 5.1 are the following:*

```
@0{scheduler.D}, @3{scheduler.clk}, @1{scheduler.r},
    @0{process_1.period}, @1{process_1.clk}, @0{T1}, @0{scheduler.C1},
    @2{delta_t}, @1{scheduler.clk}, @0{scheduler.r}, @3{delta_t},
    @2{scheduler.r}, @0{delta_t}, @0{D1}, @0{offset1},
    @3{scheduler.r}, @3{process_1.clk}, @0{scheduler.clk},
    @2{scheduler.clk}, @0{process_1.offset}, @0{the_time},
    @0{process_1.clk}
```

     Such propagation will changes most of constraints, changing the set of constraints into the following :

```
        (0.0 >= 0.0 & (0.0 = 0.0 & (0.0 = 0.0 & (@0{C1} = @0{C1} &
            (@0{C1} > 0.0 & (4.00 = 4.00 & (4.00 = 4.00 & (4.00 = 4.00
            & (4.00 = 4.00 & (0.0 >= 0.0 & (0.0 >= 0.0 & (0.0 <= 0.0 &
            (0.0 >= 0.0 & (0.0 = 0.0 & (0.0 = 0.0 & (0.0 = 0.0 & (0.0
            = 0.0 & (0.0 >= 0 & (4.00 > 0 & (0.0 = 0.0 & (0.0 = 0.0 &
            (@0{C1} >= 0.0 & (@1{the_time} >= 0.0 & (0.0 >= 0.0 & (0.0
            <= 4.00 & (0.0 >= 0.0 & (@1{delta_t} > 0.0 & (@1{the_time}
            = 0.0 + 0.0 & (0.0 = 0.0 & (@0{C1} = @0{C1} & (0.0 = 0.0 &
            (@2{the_time} >= 0.0 & (@2{process_1.clk} >= 0.0 &
            (@2{process_1.clk} <= 4.00 & (4.00 >= 0.0 & (0.0 = 0.0 &
            (4.00 < @0{C1} & (4.00 = 4.00 & (@2{the_time} =
            @1{the_time} + @1{delta_t} & (@2{process_1.clk} = 0.0 +
            @1{delta_t} & (@0{C1} = @0{C1} & (4.00 = 0.0 + @1{delta_t}
            & (@3{the_time} = @2{the_time} + 0.0 & (@2{process_1.clk}
            = @2{process_1.clk} & (@0{C1} = @0{C1} & (4.00 = 4.00 &
            (@3{the_time} >= 0.0 & (@2{process_1.clk} >= 0.0 &
            (@2{process_1.clk} <= 4.00 & (4.00 >= 0.0 & (1.0 > 0.0 &
            (1.0 = 1 &
            TRUE))))))))))))))))))))))))))))))))))))))))))))))))))))
```

2. Unnecessary constraints elimination
   In this phase we simplify the remaining clauses by performing boolean evaluation to clauses and omitting clauses that can directly be evaluated to **True**.

   Example 5.6. *Performing unnecessary constraints elimination on the result of the previous constant parameter propagation before provide us with the result below:*

```
(@0{C1} > 0.0 &  (@0{C1} >= 0.0 & (@1{the_time} >= 0.0 &
    (@1{delta_t} > 0.0 & (@1{the_time} = 0.0 + 0.0 &
    (@2{the_time} >= 0.0 & (@2{process_1.clk} >= 0.0 &
    (@2{process_1.clk} <= 4.00 & (4.00 < @0{C1} &
    (@2{the_time} = @1{the_time} + @1{delta_t} &
    (@2{process_1.clk} = 0.0 + @1{delta_t} & (4.00 = 0.0 +
    @1{delta_t} & (@3{the_time} = @2{the_time} + 0.0 &
    (@3{the_time} >= 0.0 & (@2{process_1.clk} >= 0.0 &
    (@2{process_1.clk} <= 4.00 )))))))))))))))
```

3. Arithmetic propagation
   Next, we further simplify the clauses by performing arithmetic evaluation to all the clauses.
4. Swap propagation
   Last, we order the clauses with the following policy :
   - If the clause only has constant values and parameters, then it should be in the format of $a\,\{< \mid \leq \mid = \mid \geq \mid > P\}$ where a is a constant value and P is a parameter or a set of parameters.
   - If the clause only has constant values, parameters and other variables, then it should be in the format of $a\,\{+|-\}\,P\,\{< \mid \leq \mid = \mid \geq \mid > P\}\,v$ where a is a constant value, P is a parameter or a set of parameters, and v is a variable or a set of variables.

   The ordering policy allow us to efficiently propagate the equality of variables in the terms of constant and parameters in the next steps.

Example 5.7. *One run of the all the processes in the constraint processing phase will result in the following constraints :*

```
(@0{C1} > 0.0 & (@0{C1} >= 0.0 & (@1{the_time} >= 0.0 & (@1{delta_t} >
    0.0 & (@1{the_time} = 0.000000 & (@2{the_time} >= 0.0 &
    (@2{process_1.clk} >= 0.0 & (@2{process_1.clk} <= 4.00 & (4.00 <
    @0{C1} & (@2{the_time} = @1{the_time} + @1{delta_t} &
    (@2{process_1.clk} = @1{delta_t} & (4.00 = @1{delta_t} & (@3{the_time}
    = @2{the_time} & (@3{the_time} >= 0.0 & (@2{process_1.clk} >= 0.0 &
    (@2{process_1.clk} <= 4.00 )))))))))))))))
```

We then iterate the clauses through the above four processes until there is no possible value / parameter propagation that can be done and no more boolean and arithmetical evaluation that can be done to simplify the clauses.

Example 5.8. *After several iterations we obtain the following as the final result of our process constraint to the problem in example 5.1:*

```
(@0{C1} > 0.0 & (@0{C1} >= 0.0 & (4.00 < @0{C1} )))
```

- Existelim approach

  Mathsat5 developer provides the function existelim to do quantifier elimination on the constraints, both can be used interchangeably. This approach offers the power of Fourier-Motzkin elimination on cases where there is no assignment to a certain variables, i.e. on variable x there are only constraints involving $<$ and $\leq$ inequalities. In this case the previous approach does not work as it only works for existential elimination through equality propagation. For problems involving states where guards can be taken anytime within certain range, i.e the guard is not in the form $x = a$, the existelim approach is advised.

After processing The above constraint processing is then followed through by the constraint filtering phase. In this phase we perform redundant elimination process to eliminate :

1. Occurrence of the same constraint more than once This can happen from inference from different parts of constraints that end up to the exact same constraint, or same constraint with different formulation
2. Constraints that are already implied by other constraints For example several constraints might lead to a region smaller than the region implied by constraint A. In this case we only need to retain the constraint with bigger region that includes others, and omit the already implied constraints.
3. Constraints that evaluates to true.

Last we perform the string constraint processing to present the final constraints in readable format.

Example 5.9. *In the end of all the schedulability analysis process this is the result that we obtain for example 5.1:*

```
sensitivity constraint string = !((!(C1 <= 4) & !(C1 <= 0)))
```

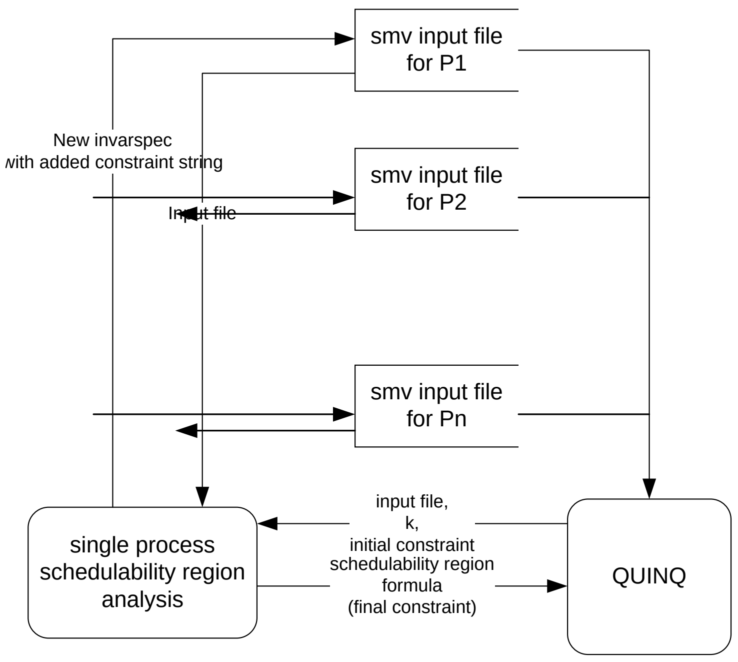

smv input file for P1
New invarspec with added constraint string
smv input file for P2
Input file
smv input file for Pn
input file, k, initial constraint
single process schedulability region analysis
schedulability region formula (final constraint)
QUINQ

Figure 5.9: A flow for the feasibility analysis implementation for n- processes.

5.3.3 Iterative schedulability analysis for periodic tasks

For periodic task system with hierarchical priority, to simplify the problem each time we only perform schedulability check to the scheduler for each task. Thus we need to do the previous feasibility analysis process iteratively n-times for all tasks $\tau_1 \ldots \tau_n$. The outermost loop is the iteration of schedulability region analysis to all the tasks $\tau_1, \ldots, \tau_n$ in the system. We illustrate this in Figure 5.9. On the other cases where we model everything in one file then this iteration over tasks is not necessary.

Chapter 6

Application Examples

6.1 Standard Periodic Case

In this section we report some examples of the tool Quinq and of the underlying method application on periodic task systems.

6.1.1 Simple task set

To illustrate the potential applications of the methodology in a design flow for embedded systems, we propose a couple of simple but explanatory use cases. These examples problem were presented in Chapter 2 as motivational example scenarios. The complete results shown below have been obtained automatically with Quinq.

In both examples we use a periodic task system $\mathcal{S}_1 = \{\tau_1, \tau_2\}$. Periods are fixed and chosen equal to the relative deadline: $\mathbf{T_1} = \mathbf{D_1} = 20$ and $\mathbf{T_2} = \mathbf{D_2} = 30$.

Use case 1: For this example, we consider a design scenario in which the computation times are not known with a sufficient precision and the system is very close to the border of the schedulability region found with traditional real-time scheduling theory (i.e., assuming null offset). The design problem is how to choose a positive offset O_2 so as to maximize the schedulability region (and hence the robustness of the system). To this end we can compute the schedulability region with respect to the following set of free parameters: $\{C_1,\, C_2,\, O_2\}$ fixing $O_1 = 0$. The tool, after some iteration, produces the following symbolic expression for the feasibility region:

$$\begin{array}{ccccccc}
(C_1 < 20) & \wedge & & & & & \\
(C_1 + C_2 < 30) & \wedge & & & & & \\
(C_1 + C_2 < 20 & \vee & 2C_1 + C_2 < 30 + O_2) & \wedge & & & \\
(2C_1 + C_2 < 40 & \vee & 3C_1 + C_2 < 50) & \wedge & & & \\
(C_2 < 20 - O_2 & \vee & C_1 + C_2 < 40 - O_2 & \vee & 2C_1 + C_2 < 30) & \wedge & \\
(C_1 + C_2 < 20 & \vee & 3C_1 + 2C_2 < 60 & \vee & 4C_1 + 2C_2 < 60 + O_2) & \wedge & \\
(C_1 + C_2 < 20 & \vee & 2C_1 + C_2 < 30) & & & &
\end{array}$$

In Figure 6.1, we can see three different sections obtained cutting the three-dimensional schedulability regions with the planes $O_2 = 0$, $O_2 = 5$, and $O_2 = 8$. In accordance with the findings of real-time scheduling theory, $O_2 = 0$ corresponds to the smallest schedulability region. The choice $O_2 = 8$ is the one that ensures the maximum robustness for the choice of C_1, while O_2 corresponds to the greatest schedulability region.

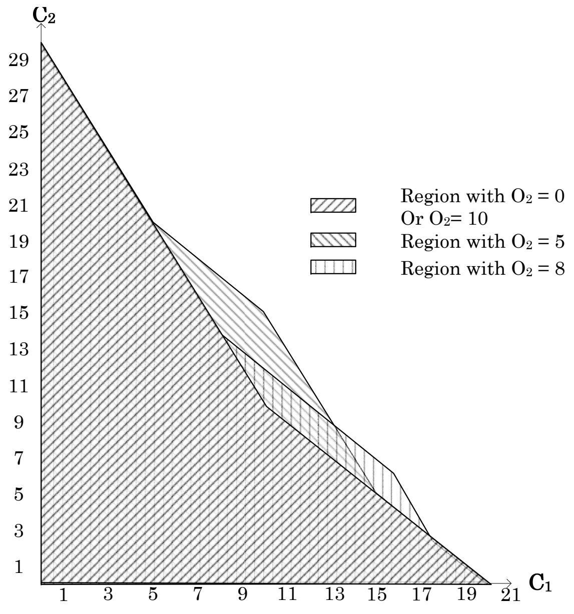


Figure 6.1: Feasibility regions in the domain of C_1 and C_2 for different values of O_2

Use case 2: In the next example, we consider that the worst computation time of the tasks are fixed, $\mathbf{C_1} = 11$ and $\mathbf{C_2} = 12$. This task system is not schedulable with zero offsets as it is possible to see with a standard response time analysis. Therefore, our purpose is to identify the possible values for the offsets O_1 and O_2 that make the system schedulable.

Using the tool, we computed the feasibility region, in the domain $O_1 \times O_2$, expressed in the constraints below:

$$\begin{array}{r}(O_2 < O_1 + 7 \vee O_2 > O_1 + 4)\wedge \\ (O_2 < O_1 - 3 \vee O_2 > O_1 - 6)\wedge \\ (O_2 < O_1 + 17 \vee O_2 > O_1 + 14)\wedge \\ (O_2 < O_1 - 13 \vee O_2 > O_1 - 16)\wedge \\ \dots\end{array}$$

The feasibility region for this case is not even connected, as it consists of parallel stripes as shown in Figure 6.2. This is a perfectly natural result since by increasing the offsets, we periodically get the same system (except for an initial transient). Therefore, the procedure can converge only because we pose bounds for the search in the parameter space ($0 \leq O1 \leq 17$, $0 \leq O2 \leq 20$).

Using those constraints we can simply pick the values for O_1, O_2 and O_3 inside the feasibility region to make $\mathcal{S}_2$ schedulable. We can choose $O_1 = 5$ and $O_2 = 1$ for example. And the system will now be schedulable with the system utility 95%.

6.1.2 Large task set

For an example of a large task set that the tool could handle we present a system with 10 periodic tasks. The configuration values for all the fixed variables in the system is given in Table 6.1.2. The parameters are $C1$, $C2$, $C3$, $offset1$, and $offset2$.

In this example case we are working in 5 dimensional parameter space. The result that we obtain for

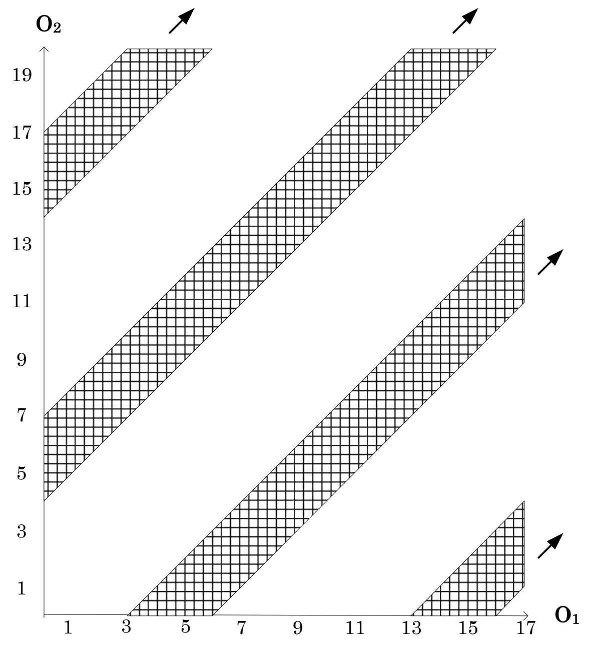


Figure 6.2: Regions of feasibility in the domain of O1 and O2.

the schedulability region of the system consists of 719 constraints. Due to the dimension of the parameter space and the large number of constraints we obtained as the result, here we will only show one example of cut of the final region in chosen 2 dimensions.

Given that $offset1 = 6$, $offset2 = 5$, we draw the region for $C3 = 3$, $C3 = 4$, and $C3 = 5$ in $C1$ and $C2$ dimension in the Figure 6.3. The result shows increasingly small region of feasibility with increasing value of $C3$.

6.2 Arbitrary real–time system case

To show the usage of Quinq in a real–time system with arbitrary activation pattern and scheduling policy we present an example case on a distributed heterogeneous system [39]. This example has also been presented as motivational scenario in Chapter 2.

6.2.1 System description

We have taken as a case study a simplified version of a Heterogeneous Communication System (HCS), such as one that could be found on board of aircrafts [27]. The architecture of the system is shown in Figure 6.4. The HCS system contains a common server, wired and wireless communication networks and a number of devices. The components of the network communicate through Network Access Controllers (NAC), which perform gateway function and routing, and are connected in a daisy chain topology. In this case study we focus on audio devices, which are required to distribute music and audio announcements to the main cabin. The audio stream is transmitted by the server through the network. To avoid echo effects, the devices must reproduce the audio at synchronized instants. For this reason, the network, server and devices implement also the Precision Time Protocol (PTP) [47] to synchronize their clocks.

Task	Period	Deadline	Computation time	Offset
1	15	15	$C1$	$offset1$
2	20	20	$C2$	$offset2$
3	25	25	$C3$	8
4	30	30	2	8
5	40	40	2	2
6	50	50	2	4
7	60	60	2	7
8	75	75	2	7
9	80	80	2	10
10	100	100	2	8

Table 6.1: The configuration of an example large periodic task system of 10 tasks with 5 parameters

The communication between the server and device is asynchronous. The server sends an audio packet every $audPeriod$ ms. Audio packets are characterized by two parameters: a sequence number i and a timestamp ti denoting the time the packet has to be played at the device. Due to varying network conditions, packets arrive at the device (except if they are lost) with a minimal latency $Lmin$ and a maximal latency $Lmax$. The NACs simply forward the incoming packets to the devices. Packets passing through the NACs experience delay of $Lnac$ during which they are preprocessed by the NACs. The device processing time for each audio packet is τ, after which the device is ready to receive the next audio packet. The PTP protocol runs on the server, the devices and NACs, and is used to synchronize the respective clocks. Figure 6.5 depicts the message sequence of clock synchronization between a device (slave) and the master clock. Various timing delays are to be guaranteed. For example, in a scenario where two devices are connected to the server, it should be guaranteed that both devices are synchronized within an error of 0.1 ms (synchronization precision).

Our objective is to identify the largest region of the parameter space in which the the correct functioning of audio streaming and clock synchronization can be guaranteed. To do so, we employ feasibility analysis through parametric timed automata offered in Quinq. However, HCS is too complex to be parametrically-modelled completely. Therefore, some of the above requirements have to be relaxed before we attempt to parametrically model the system.

The simplified system contains only one server and many NACs and devices where one NAC may be associated to at most one device and one other NAC. Also, only the PTP parts on the server and devices are modelled. Additionally, every packet will have to experience the maximal delay $Lmax$ when traversing the medium. Furthermore, we only partially model the unreliability of the network. The system high-level description can be viewed logically as in Figure 6.6. Last, the transmission priority of PTP messages are assumed to be higher than that of audio packets, however, an ongoing transmission of an audio packet will not be preempted by a PTP message.

A complete set of models for this simplified system was developed in UPPAAL [37] as a network of 13 timed automata [38]. In UPPAAL, HCS is modelled as a network of extended timed automata with global real-valued clocks and integer variables. Clock value retrieval which is essential to the PTP protocol is not supported by UPPAAL, hence the introduction of integer clocks. There are four error states in the automata network, corresponding to error conditions. One is reached when an audio packet arrives after its time-to-play. Three others are reached when buffer overruns occur in the audio buffer,

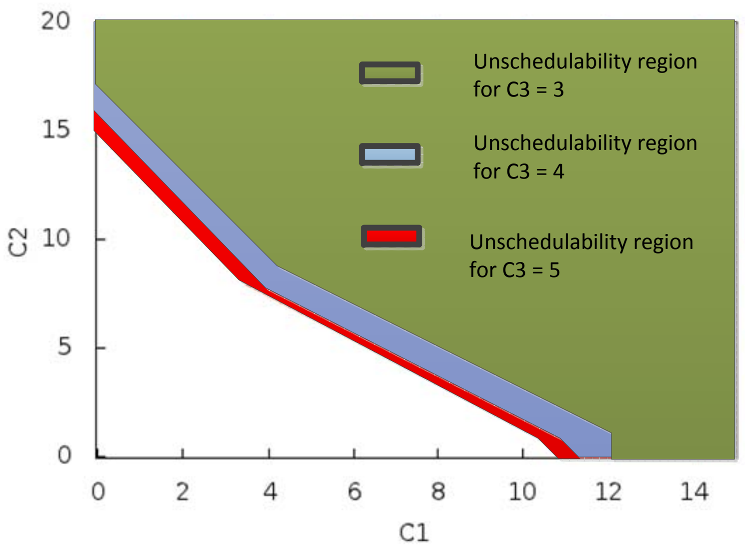


Figure 6.3: Feasibility regions in the domain of C_1 and C_2 for different values of $C3$

the NAC input buffer and the NAC output buffer. To verify that the system is schedulable, we must show that these four error states are never reachable.

6.2.2 System modelling

Because of its considerable complexity, we have worked out an abstraction of the system to limit the state space and to concentrate in isolation on each outstanding issue (the non preemptive scheduler, the different criticality of the timing constraints and so on). Our strategy is to first analyse the timing constraints of the audio stream and PTP, under certain assumptions on the PTP synchronization accuracy, and determine the acceptable clock drift relative to other parameters. In a second step, we will analyse the PTP operation to ensure that the synchronization accuracy meets the constraints defined in the first phase.

Abstract Models

We divide the abstract model of the system into two parts. The first corresponds the release of the packets on the network according to a periodic pattern. The second models the network and device, including the scheduling policy and the real-time constraints. For the latter, we have considered both the classic hard real-time constraints and firm real-time constraints.

We model the release of packets as activation automata, shown in Figure 6.7. Each stream of packets is characterized by the offset for the first release (transition from initial state to the second state), and is then periodic afterwards (self transition on the second state). A release signal is emitted every time a transition is taken, and is used to synchronize the automaton with the rest of the system. In the following we will refer to these automata also as “tasks”.

The remaining part of the system is modelled as a set of schedulability checkers [21]. Unlike in standard periodic task system work we presented before, the checkers for the two tasks are modified to model a *non-preemptive* scheduling algorithm, i.e., a transmission will not be interrupted if it has already

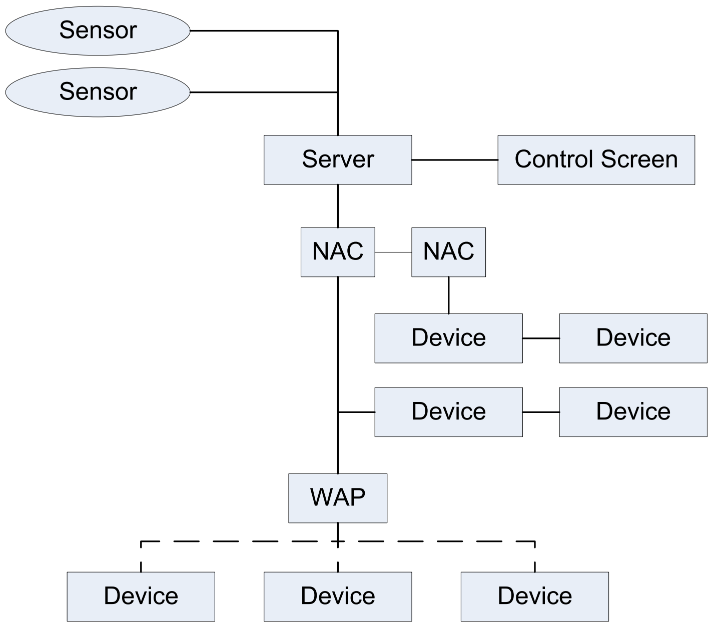


Figure 6.4: Heterogeneous Communication System (HCS)

started. The scheduler is also *prioritized*, so that when there is no ongoing transmission and many packets are ready, the PTP packets go first and the audio packets back off.

The scheduler checker for PTP packets is shown in Figure 6.8. D_1 is the deadline of PTP packets, which is less than or equal to the PTP period, while C_1 and C_2 are the transmission time of PTP and audio packets, respectively. The PTP task has a higher priority than the Audio task as specified in Section 6.2.1. In addition, the execution time of the former accounts for the total PTP load that the devices could bear and that of the latter accounts for the total delay of traversing through the medium and the NACs of audio packets. Furthermore, we have five additional variables in our system:

- $task$ denotes the currently-executed task (i.e., the current on-going transmission)
- n_1 and n_2 record the number of PTP and audio packets released during the current execution
- c is a clock accumulating the time since the task queues were last idle and
- r is a data variable used to sum up the time needed to complete all tasks released since the checker was last idle.

The transitions of the checker are intuitively interpreted as follows:

- The transitions to Idle are taken when the task instance being checked in Check or a sequence of tasks arrived in Busy, has finished execution.
- The transitions to Busy are taken when an instance of task PTP or Audio is released. Self-loops are taken to queue the newly-released instances and to retrieve them when the current execution has finished.

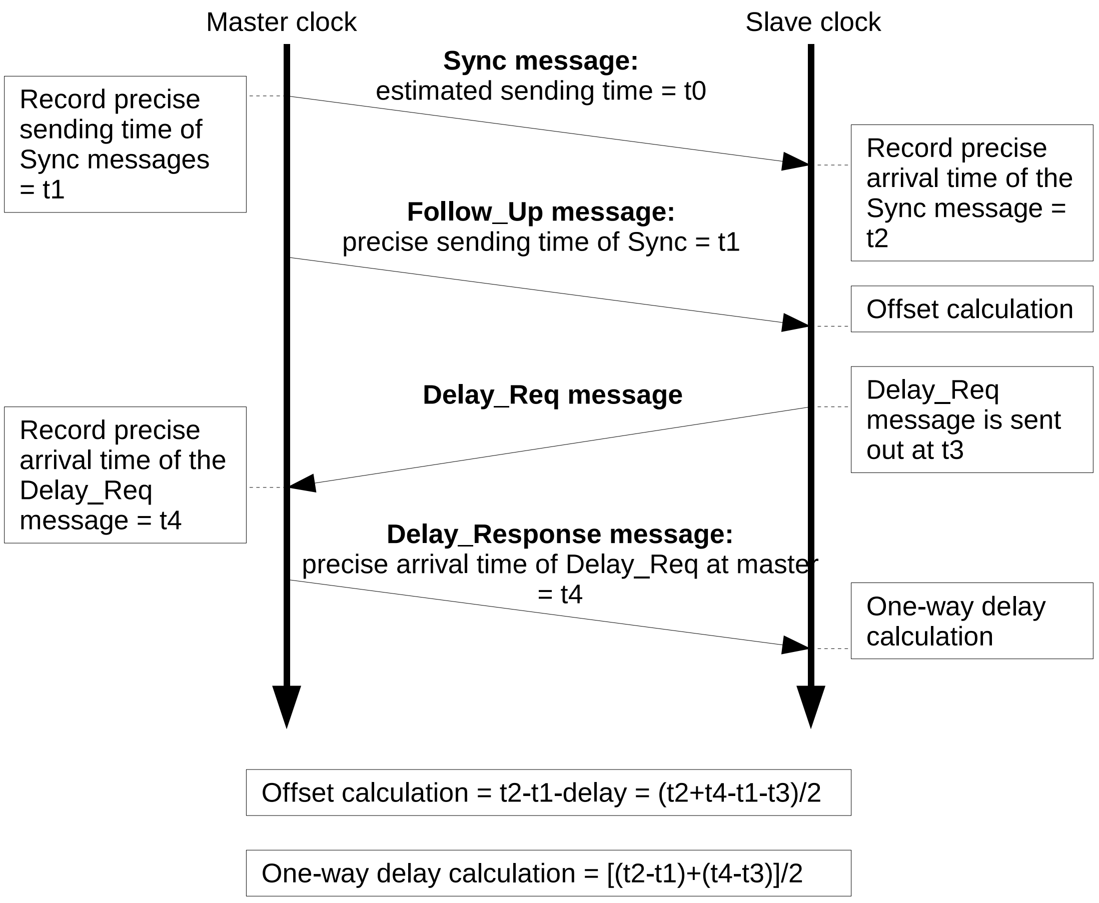

Master clock
Slave clock
Record precise sending time of Sync messages = t1
Sync message: estimated sending time = t0
Record precise arrival time of the Sync message = t2
Follow_Up message: precise sending time of Sync = t1
Offset calculation
Delay_Req message
Delay_Req message is sent out at t3
Record precise arrival time of the Delay_Req message = t4
Delay_Response message: precise arrival time of Delay_Req at master = t4
One-way delay calculation
Offset calculation = t2-t1-delay = (t2+t4-t1-t3)/2
One-way delay calculation = [(t2-t1)+(t4-t3)]/2

Figure 6.5: Time sequence diagram of message exchanged between master and slave to achieve clock synchronization

- The transitions to Check are taken when a PTP instance is (non-deterministically) chosen for checking. Before verifying the deadline, the execution time of all other PTP instances in the queue must be taken into account as they would be scheduled before the current instance, that is r should be updated to $r + C_1 + C_1 * n_1 - c$, or $r + C_1 + C_1 * n_1 - C_2$. New PTP and Audio instances in Check are ignored as they are already considered in location Busy.
- The transition to Error is taken when the currently-executed instance misses its deadline.

The scheduler checker for audio packets is shown in Figure 6.9. The Audio checker is similar to but simpler than the PTP checker because task audio has a lower priority. D_2 is the relative deadline of audio packets and Δ is introduced to account for the offset time of the local clock compared to the server clock. The worst case happens when the local clock is substantially slower than the server clock and thus when an audio packet is received, the actual deadline to be verified would be $D_2 - \Delta$ instead of D_2.

In fact, the requirement of no deadline miss (hard deadline) is difficult to obtain in real-time environments. Therefore, in order to make the analysis more practical, the requirement is relaxed by allowing an audio packet to sometimes miss its deadline (*firm real-time constraint*). A firm real-time constraint is given by a deadline and by a couple (m, n) meaning that m deadlines can be missed every n jobs. [43]. In our case study, a packet may miss its deadline as long as the previous packet has not already missed

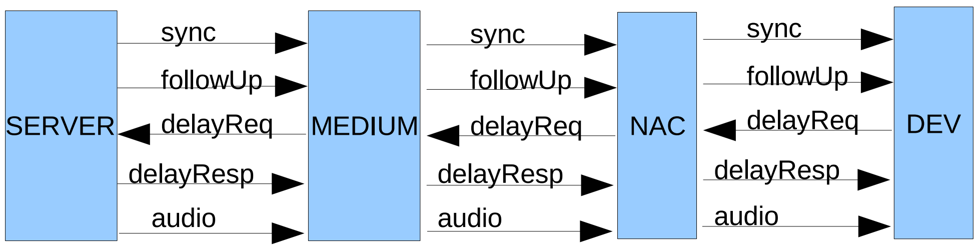


Figure 6.6: Logical model of HCS

	Experiment 1	Experiment 2
$ptpOff$	0	5
$ptpPeriod$	40	40
D_1	10	10
$audOff$	0	0
$audPeriod$	10	10
D_2	10	10

Table 6.2: Fixed parameter values in the two experiments

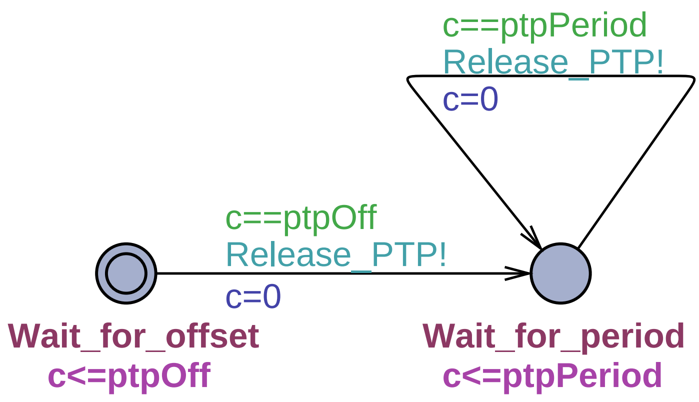
the deadline ($m = 1, n = 2$). The checker adapted for this new requirement is shown in Figure 6.10. We introduce four new variables: dm is a boolean variable used to capture the fact that one deadline miss has already happened ($dm = true$), r_1 is a real variable used to record the total execution time of all PTP instances released after the currently-checked instance and before a deadline miss or the next audio arrival, r_2 marks the next audio arrival whose time is marked in t.

Transitions entering Check1 from Idle or Busy are taken when an audio instance is (non-deterministically) chosen for checking. If another PTP instance is already released, the PTP will be executed first ($r = r + C_1$). The self-loops in Check1 are taken to accumulate the execution time of all PTP instances released afterwards until another audio instance is released or the current Audio instance misses its deadline. If one deadline miss had happened before ($dm = true$), the location *Error* is reached because of two successive deadline misses. Else, if another audio instance has already been released ($r_2 > 0$), the transition from Check1 to Check2 is taken in order to verify if the next deadline is missed again. Otherwise, the variable dm is updated to $true$ and the transition from Check1 to Busy is taken to tolerate the first deadline miss.

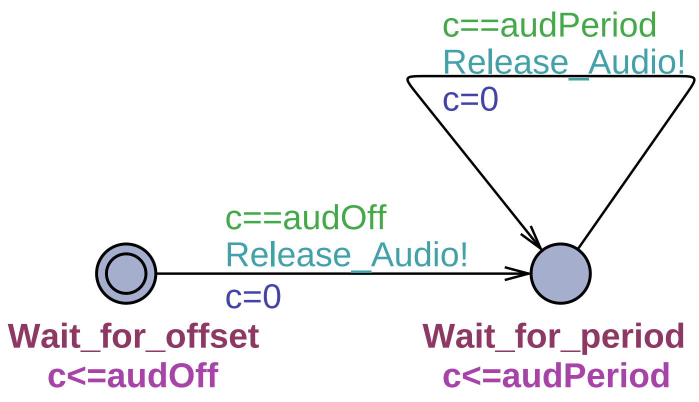
6.2.3 Example result

We have performed several experiments to compute the feasibility region of the system under a diverse set of parameters. We present here the results of two such experiments, which differ in the amount of offset by which packets are issued to the network. We have chosen as free parameters the transmission times C_1 and C_2 and the desired accuracy Δ (or drift) of the clocks ($\Delta = 0$ corresponds to infinite accuracy), given by the application of the PTP protocol. The values of the fixed parameters for each of the experiments are shown in Table 6.2. The free parameters, on the other hand, are constrained to vary in the following intervals:

$$0 < C_1 \le D_1, \qquad 0 < C_2 \le D_2$$
$$0 \le \Delta$$



(a)



(b)

Figure 6.7: PTP and audio task activation automata

The PTP checker generally runs much slower than the Audio checker which may be because the path leading to the error state of the former is generally longer than that of the latter.

Audio feasibility and error regions

We perform experiments on both Audio hard-deadline and firm-deadline checkers. Figure 6.11 and 6.12 graphically show the Audio feasibility (not shaded) and error (shaded) regions for $\Delta = 0, 3, 5, 7$ in two experiments. The feasibility regions are expressed by set of constraints, for example, with $\Delta = 5$ the feasibility region for the Audio firm-deadline checker in Experiment 1 is expressed by a conjunction of three constraints as shown below. A first evident result is that the relaxation of timing constraints pro-

```
1: $\neg[(20 < C_1 + 2C_2) \wedge (0 < C_1 \leq 10) \wedge (0 < C_2 \leq 10)] \wedge$
2: $\neg[(15 < C_1 + 2C_2) \wedge (10 < C_1 + C_2) \wedge (0 < C_1 \leq 10) \wedge (0 < C_2 \leq 10)] \wedge$
3: $\neg[(C_1 + C_2 \leq 10) \wedge (0 < C_1 \leq 10) \wedge (5 < C_2 \leq 10)]$
```

duces a larger feasibility region. This rather intuitive fact can be exactly quantified and even pictorially represented thanks to our methodology. The second insight offered by this analysis is to quantify the impact of the synchronization accuracy on the correctness of the system. Indeed, the feasibility region

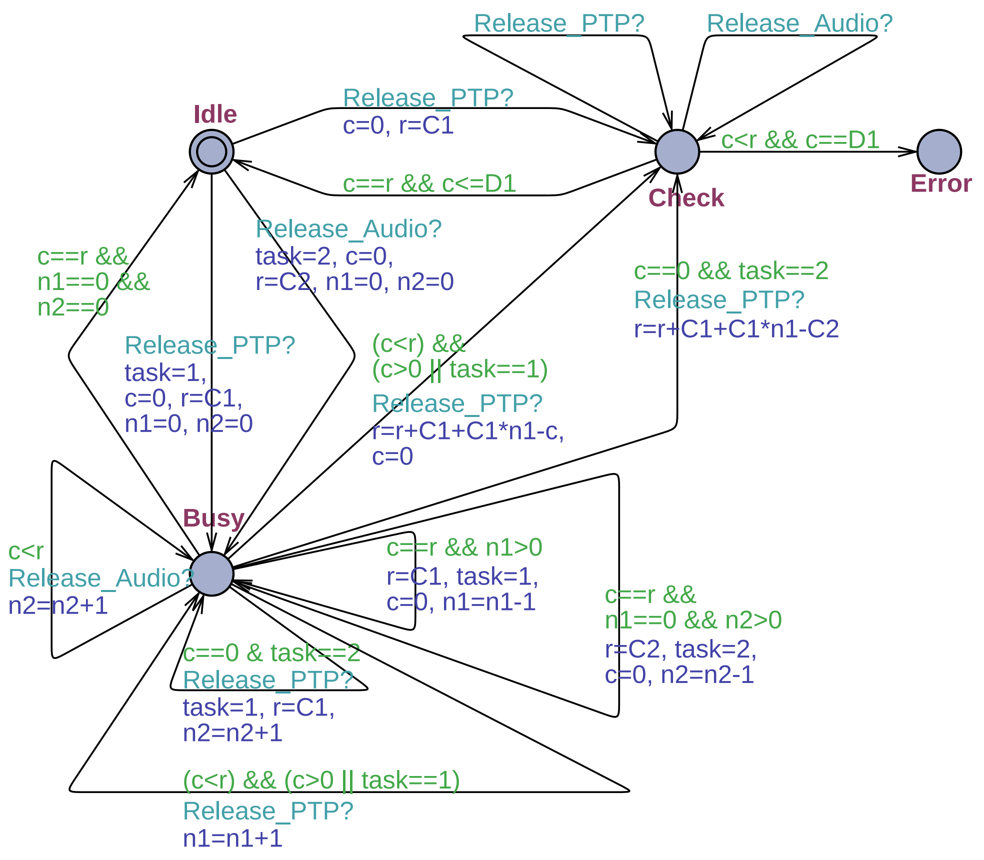


Figure 6.8: Schedulability checker for PTP packets

shrinks as Δ increases and the temporal behaviour of the system can easily be jeopardized. This result can be used as a specification for the PTP. Finally, our analysis can be used to guide the choice of activation offsets, an important degree of freedom to simplify schedulability of highly loaded systems.

PTP feasibility and error regions

The PTP feasibility region in the two experiments is also expressed in terms of sets of constraints. Figure 6.13 graphically shows the PTP feasibility (not shaded) and error regions (shaded) in the two experiments. By joining the PTP and Audio feasibility regions together, we can obtain the final region in which the whole system is guaranteed to work properly. For example, the feasibility region for the whole system in Experiment 1 is actually that for task Audio. If the device clock does not drift ($\Delta = 0$), point ($C_1 = 10, C_2 = 5$) is a schedulable point because the first PTP released at time 0 does not miss its deadline ($C_1 = D_1 = 10$), then the first Audio deadline miss happens ($C_1 + C_2 > D_2$) but the second Audio deadline is respected ($C_1 + 2 * C_2 = 2 * D_2$). Other Audio instances released at time 20 and 30 are not preempted, thus able to meet their deadlines. At time 40, the task arrival pattern is repeated exactly the same as time 0. However, point ($C_1 = 10, C_2 = 5$) is no longer a feasible point when $\Delta = 5$. Because the first Audio instance misses its deadline as it does when $\Delta = 0$ and so does the second Audio instance. So, point ($C_1 = 10, C_2 = 5$) should be in the error region which is verified easily by looking at Figure 6.11(b).

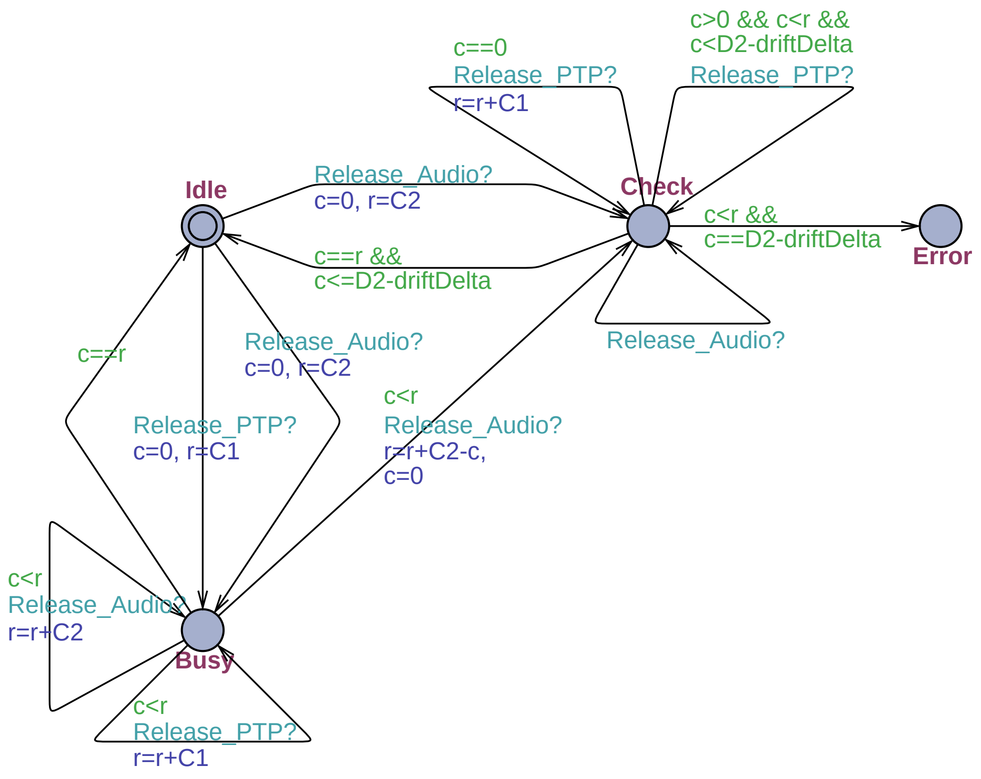


Figure 6.9: Schedulability checker for audio packets (hard deadline)

6.2.4 Example search optimization result

On the case where there are many possible traces on the system, and they consist of long traces, the search optimization approach of using numeric model checker such as UPPAAL is very beneficial. Compared to the time it takes to perform BMC by itself to iterate several times over long bounded steps, the time UPPAAL needs to grid the area is very less. UPPAAL performance in finding a trace is also affected by the length of the trace. However, it is still in the order of seconds for UPPAAL as it only consider ground parameter model. The PTP experiment is a perfect example for such case.

As an illustration, Figure 6.14 shows the different result region of PTP experiment within different time frame. Using symbolic search, it took us 30 minutes to find a region in the shape of a line. However, using search optimization with UPPAAL non-parametric search we obtain ourselves 7 regions that cover almost all of the final region just within 5 minutes.

While this result benefits from UPPAAL, we still suffer from UPPAAL's problem with granularity. We can only obtain trace guide from discrete points in UPPAAL. To be able to get more traces we need to increase the granularity of the UPPAAL model as to cover the traces resulting from points in the in between regions.

6.2.5 Discussion

Through this example we have shown how to apply our method to a distributed Heterogeneous Communication System (HCS). We show that our method could also account for non-preemptive scheduling and soft real-time constraints. In addition, we have shown how to use this technique in the context of a

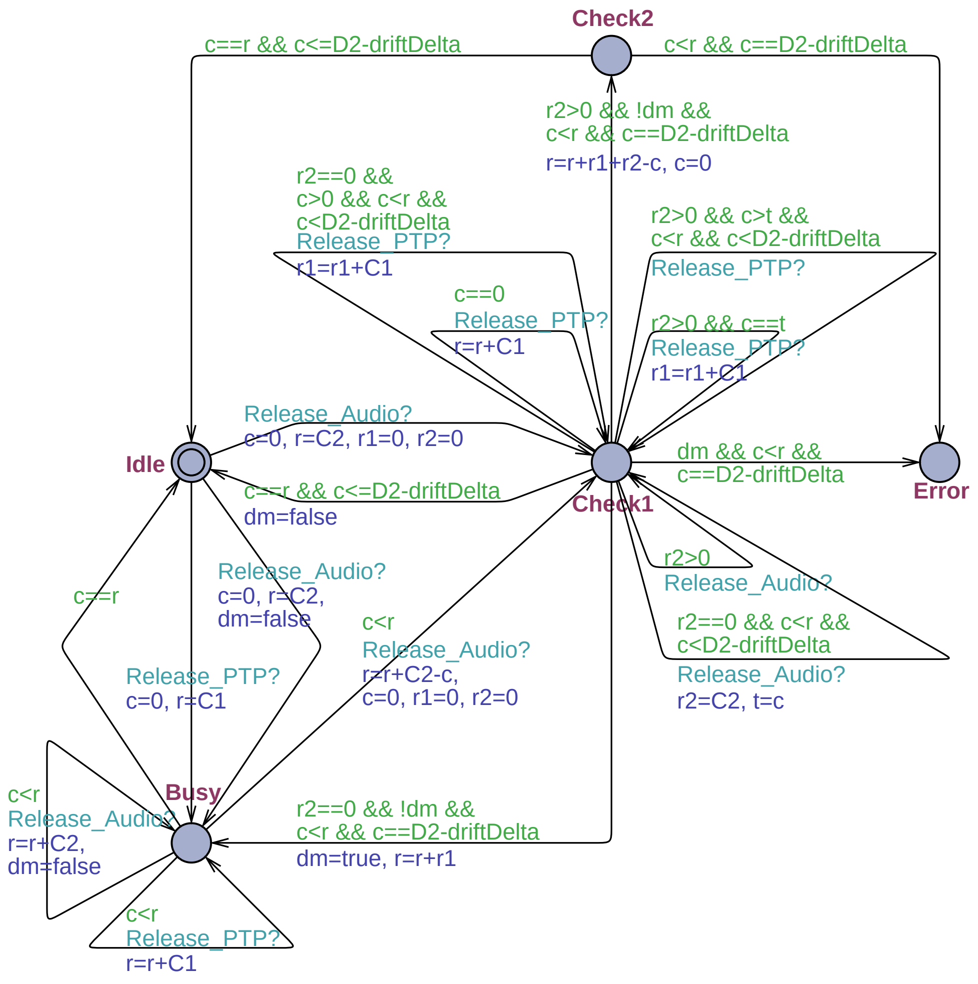
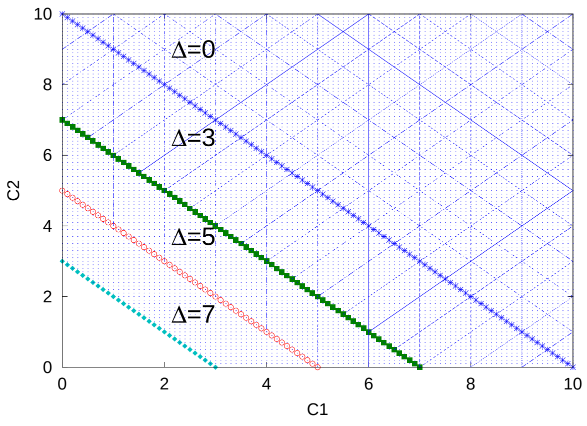
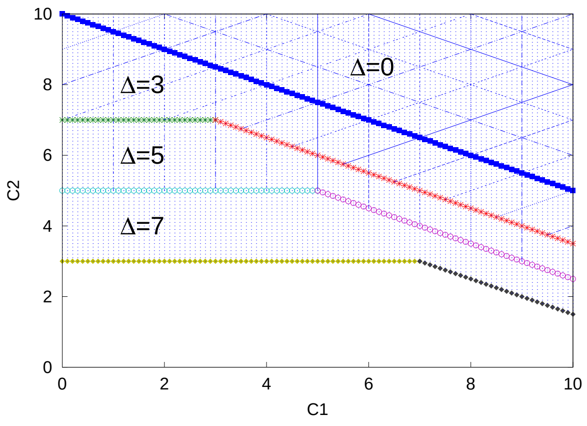


Figure 6.10: Schedulability checker for audio packets (firm deadline)

network, rather than the more traditional processor scheduling problem. Starting from the full specification, we have derived a simplified model which still exhibits the required characteristics. The analysis identifies non-trivial feasible regions for various values of the desired synchronization accuracy.

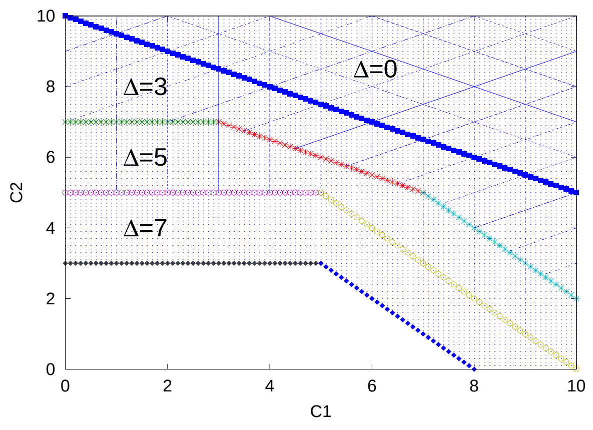


(a) Hard deadline

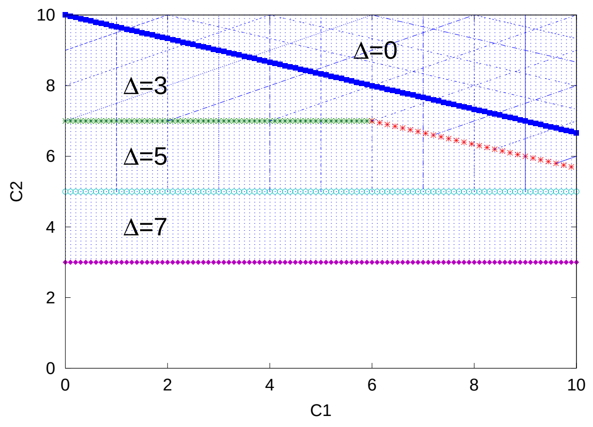


(b) Firm deadline

Figure 6.11: Audio feasibility and error regions for $\Delta = 0, 3, 5, 7$ in Experiment 1

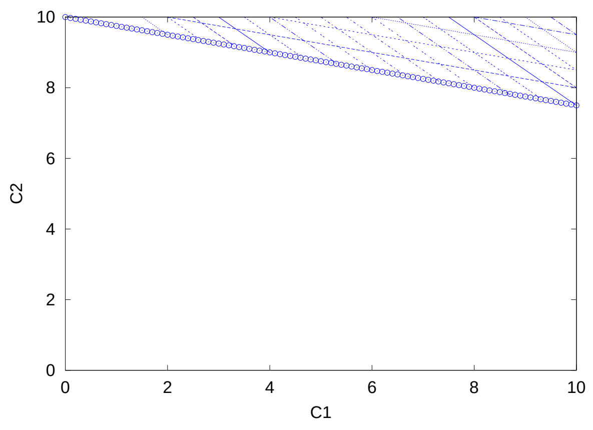


(a) Hard deadline

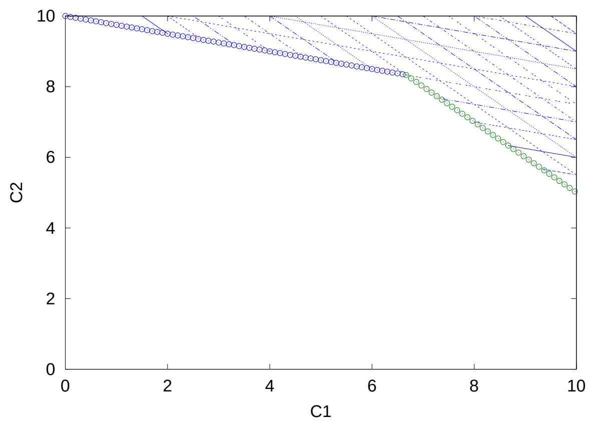


(b) Firm deadline

Figure 6.12: Audio feasibility and error regions for $\Delta = 0, 3, 5, 7$ in Experiment 2



(a) Experiment 1



(b) Experiment 2

Figure 6.13: The PTP feasibility and error regions in two experiments

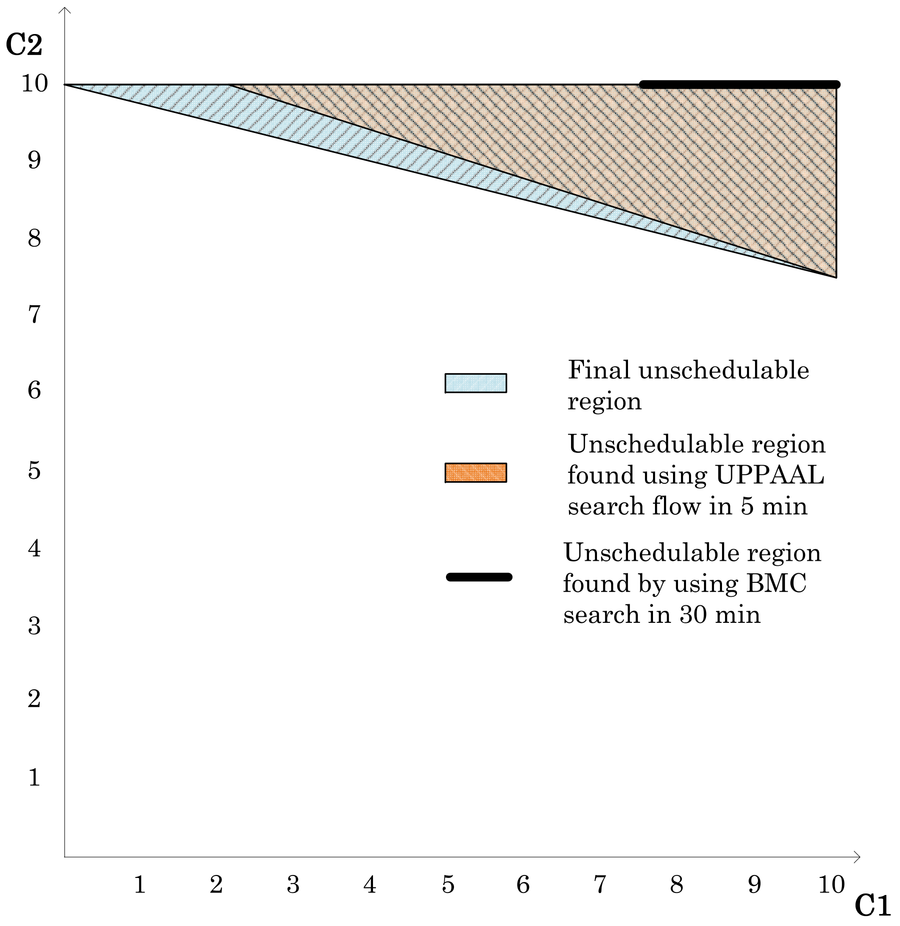


Figure 6.14: Comparison of regions found using Quinq with BMC search trace and search optimization method using UPPAAL

Chapter 7

MPA PTA

In this chapter we present the other possible application for parametric timed automata feasibility analysis in general. In particular, here we show how our method can be used to complement other approach in real–time system perfomance analysis[51].

7.1 Motivation

System level analysis plays an essential role in the design of hard real-time embedded systems at early stage. Several different methodologies that address the problem of performance analysis of different embedded systems have been presented recently. These methodologies are often based on different forms of abstraction, however they can be applied to a specific or limited set of system architectures or parameter space. For example the Modular Performance Analysis Toolbox [53], based on the Real-Time Calculus (MPA-RTC) [52], makes use of functions on the time-interval domain to represent both system workload and availability of computation and communication resources. Component interaction is modelled by sets of functions, instead of streams of signals, tokens or other activity triggers. As this features compositionality also on the level of the formal analysis, MPA-RTC-based supports a wide and efficient performance evaluation of key metrics of component-based real-time systems. Pessimism as contained in the obtained guarantees w. r. t. worst- and best-case system behaviours, e. g. burst-sizes, backlog-sizes and delays, can be avoided as long as the system under analysis matches the model of computation inherent to RTC. However, it may be difficult to adequately model components exhibiting state-dependent behaviours, e. g. CPUs whose speed changes with the size of the backlog buffer. For less pessimistic results in such situations, [35, 36] developed a methodology for embedding Timed Automata (TA) [2] based component models into a MPA-RTC-based system model. However, with this approach it is possible to do schedulability analysis only for a fixed set of parameters, e. g. a fixed CPU speed and fixed buffer sizes, while a big space of parameters remains unstudied. In this work, we approach this issue by coupling parametric TA and MPA-RTC-based component models and thereby making the Parametric Schedulability Analysis tool [21] accessible for an MPA-RTC-driven system analysis (cf. Figure 7.2).

In this example we present an embedding of Parametric Schedulability Analysis into a MPA-RTC-driven performance evaluation. Our goal is to enlarge the design space that can be explored by joining the capabilities of design methodologies based on different concepts of abstraction: On one hand the proposed methodology limits state space explosion as intrinsic to formal verification, including the Parametric Feasibility Analysis in Quinq, to the level of isolated (sub)-components, rather than experiencing it when analysing the complete overall system model. Because we emphasize the use of the MPA-RTC-driven performance analysis methodology for analysing complete systems, state-based analysis is limited to individual components only. On the other hand, the obtained performance metrics for overall system designs may not be (destructively) pessimistic, since one applies pure RTC-based analysis only for those system components which can be tightly abstracted using analytic RTC models. Contrary to existing couplings of methodologies, the proposed framework features the finding of regions of parameter values for which a system maintains its worst-case guarantees w. r. t. backlog-sizes and delays.

7.2 Real–Time Calculus

Real-time Calculus (RTC) [12, 52] yields an analytic methodology for compositionally obtaining key performance measures of systems with real-time constraints. In this work we exploit the pattern presented in [35, 36] for embedding TA-based model descriptions [2] into RTC-driven performance analysis. However, contrary to the existing work, the usage of PTA proposed here allows one to obtain regions of RTC-curve parameters rather than individual values. This of course holds also for the features of the embedded component model, as regions on bounds on execution speeds, buffer sizes etc. can be obtained, rather than point-estimates.

In this section we present a hybrid methodology that unifies two different approaches to complex real-time system analysis. These two approaches share the view of the system as a composition of computational and communication components (resources). Tasks are executed on the computational elements and data packets are sent through the communication element. The order of task execution or data transmission is defined by a chosen scheduling policy. Further, we will not distinguish the computational and communicational element as well as tasks and data packets.

Each resource acts as a Greedy Processing Component (GPC) which means that it is never idle unless there are no more tasks to be scheduled. The task activation, whether it is processed directly or added to the scheduling buffer for further processing, is marked with a start event, while the end of the task execution is marked with an end event. Timed event traces (see Section 7.2) are used to represent both the input task activation pattern of a component and the output pattern, which in multi-component systems may become an input pattern to the following component. The timed traces abstract characterization by a pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$ of arrival curves is used in the analytic approach while Timed Automata (TA) forms the foundation of the state-based model description.

In the following we summarize some aspects of the theory that is decisive for this work:

Streams A timed event is a pair (t, τ) where τ is some event label or type and $t \in \mathbb{R}_{\geq 0}$ some non-negative time stamp. A timed trace $tr := (t_1, \tau); (t_2, \tau); \ldots$ is a sequence of timed τ-events ordered by increasing time stamps, s. t. $t_i \leq t_{i+1}$ for $i \in \mathbb{N}$ and $\tau \in Act$ with Act as a set of event labels or event types.

RTC-curves for representing streams Let tr be a stream, which is a trace filtered w. r. t. to some event type. A stream can be abstractly characterized by a pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$, which is a pair of so called arrival curves [52]. These arrival curves bound the number of events of the respective type, seen on tr for any interval $\Delta \in [0, \infty)$. Let $R(t)$ denote the number of events that arrived on the stream tr in the time interval $[0, t)$, upper and lower arrival curve satisfy the following equation

$$\alpha_{in}^{low}(t-s) \leq R(t) - R(s) \leq \alpha_{in}^{up}(t-s) \text{ with } 0 \leq s \leq t. \tag{7.1}$$

As each event from a stream may trigger behaviour within a component, arrival curves provide abstract lower and upper bounding functions α_{in}^{low} and α_{in}^{up} w. r. t. the resource demand experienced within time interval $\Delta := t - s$. Furthermore, one may note that for a given pair α_{in}^{low} and α_{in}^{up} there might be a (possibly infinite) set of streams of computation stimuli, namely all streams whose counting function R satisfies (7.1). For simplicity, we denote the pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$ as α_{in}. A stream whose cumulative counting function satisfies (7.1) is said to be bound by α_{in}.
Arrival curves are a generic way for characterizing standard real-time traffic models ranging from strictly periodic event arrivals over PJD-streams, i. e. traces where event arrivals are periodic with jitter and a minimum distance, up to sporadic event arrivals.

RTC-based approximations This work deals with complete amounts of events only, i. e. the number of events to be processed are integer values. In RTC such scenarios are modelled by staircase curves. We can model staircase curves as set of staircase curves composed via nested minimum and maximum operations. However, for the sake of clearness the nesting of minimum and maximum operations is restricted, s. t. it is assumed that upper arrival curves are obtained as minimum and lower arrival curves as maximum on a set of staircase curves:

$$\begin{gathered}\alpha^{up}(\Delta) := \min_I(\alpha_i(\Delta)) \text{ and } \alpha^{low}(\Delta) := \max_J(0, \alpha_j(\Delta)) \text{ with} \\ \alpha_{\{i,j\}}(\Delta) := N_{\{i,j\}} + \left\lfloor \frac{\Delta}{\delta_{\{i,j\}}} \right\rfloor \end{gathered} \tag{7.2}$$

The fact that upper (and lower) RTC curve(s) are constructed by a single minimum (and maximum) operation (not nested min/max computations) reduces the complexity of embedding TA into RTC-driven performance analysis considerably.
An example of this setting is given in Figure 7.1: the upper arrival curve α^{up} is constituted by the minimum of two staircase curves. The lower curve consist of the maximum of the constant 0-function and some staircase function α_1^{low}. Parameter N_1^{up} of the upper curve α_{in}^{up} models the (burst) capacity which is the number of events which can arrive at the same instant of time in any stream bounded from above by α_{in}^{up}. Hence δ_1^{up} of the upper curve defines the minimum distance of two events once a burst had occurred. In this realm the step width δ^{low} of curve α_{in}^{low} models the maximum distance between two events in any stream bounded from below by α_{in}^{low}.

7.3 System modelling

The tools coupling flow is presented on Figure 7.2. Here we see three processing components connected in a data flow manner. Two of which, the first and the last one, are RTC-based components while the one in the middle is a TA-based component. The output curves of the first component acts as the input curves for the second one therefore they are translated to the TA-based event generator (see Section 7.3.1) for the second component. The tasks are then processed by the second components represented by a TA-based scheduler generalized for Quinq (see Section 7.3.2). The end of each task processing is marked with an output event that is then captured by the two observer automata. With observers we address a TA which guarantees the invulnerability of a fixed parameter value describing burst sizes, widths of stairs or minimum/maximum distance of events. An example of an upper observer automaton as implemented in UPPAAL is given in Figure 7.3.

In this flow we make use of the UPPAAL toolbox as a part of the MPA-RTC/Quinq coupling flow that is used as one of the MPA-RTC to Quinq translation chain links. Three TA-based models, the input event

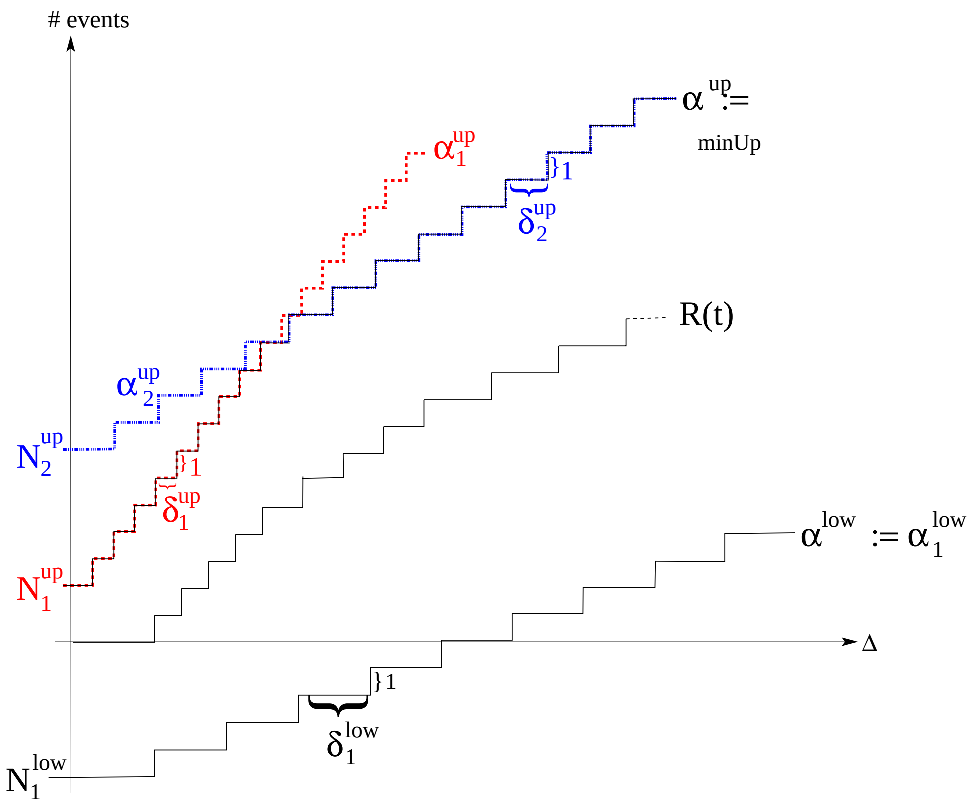


Figure 7.1: Upper and lower (staircase) arrival curves

generator, the scheduler and the two observer automata, are initially represented in UPPAAL toolbox. However, the Quinq tool requires a symbolic representation of the TA using the NuSMV language. We use the Java UPPAAL to NuSMV Translator (JUNT) developed in this work to translate the UPPAAL model of the system into NuSMV.The .XML file of UPPAAL is taken as the input to the JUNT translator. The JUNT output file (.SMV) is the executable NuSMV file that is accepted by the Quinq tool. This file is then processed by the Quinq tool and the found schedulability region of required parameters (e.g., $N_{1..k}$, $\delta_{1..k}$, N_{Obs}, δ_{Obs}) is then given back to the MPA-RTC toolbox.

While performing the simulation using the Quinq tool we can fix either parameters of the input curves ($N_{1..k}$, $\delta_{1..k}$) and find the region of the output curves parameters (N_{Obs}, δ_{Obs}) or vice versa. The analysis type performed while fixing the input curves parameters we call *Forward Analysis*. Respectively, the type of analysis performed when fixing the output parameters while searching the region for the input parameters we denote as *Backward Analysis*. It is also possible to fix both input and output characterization values in order to find an area for such parameters as the buffer size, maximum execution latency and/or max execution time which we would call as *Processing Element Analysis*. The schedulability region of the searched parameters are then used in the MPA-RTC toolbox to produce the RTC-conformant either input or output curves, depending on the analysis flow.

7.3.1 From RTC-curves to event emitting PTA

Upper and lower RTC-based staircase curves can be implemented by sets of cooperating PTA, where the emitted event signals serve as input stimuli to the down-stream PTA-based scheduler model; the latter is

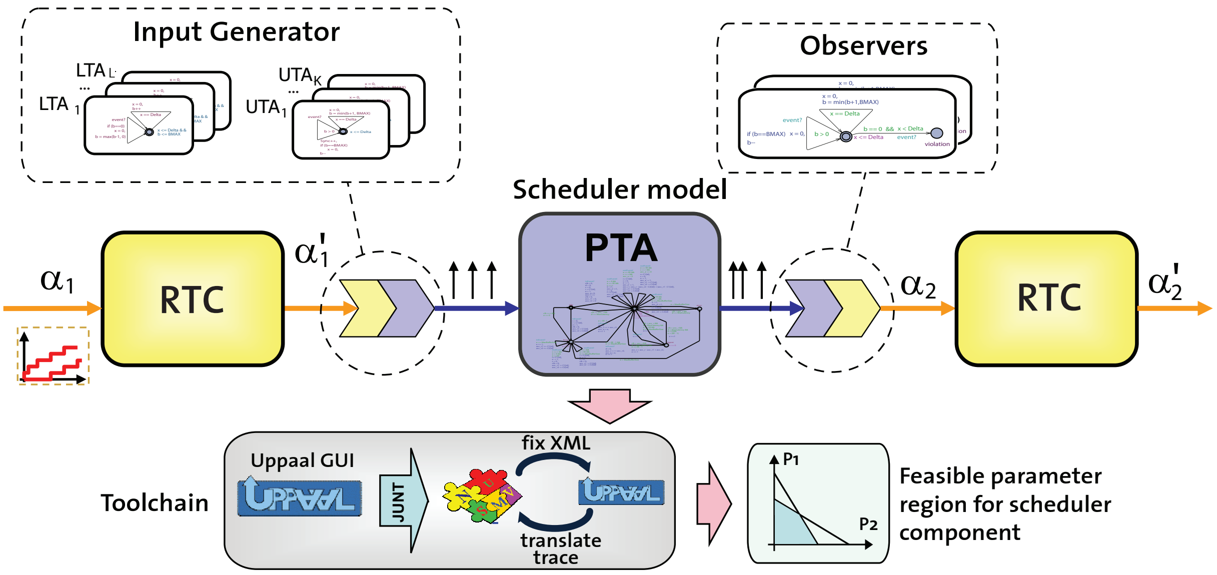


Figure 7.2: Overview of analysis approach

described in the next section. The different PTA are coordinated by making use of broadcast channels as implemented in the timed model checker UPPAAL [10].
With our approach we translate RTC staircase curves into an event-emitting PTA as follows:

1. Curve parameters $\delta_{\{i,j\}}$ and $N_{\{i,j\}}$ (see Eq. 7.2) are either chosen fixed or are in the parameter set of the PTA. Each staircase curve $\alpha_{\{i,j\}}$ (Eq. 7.2) is implemented by its own event emitting PTA, where in case of an upper curve α_i we speak of a timed automaton UTA i and in case of a lower curve α_j of a timed automaton LTA j. The generic implementation of an UTA and LTA within the timed model checker UPPAAL is shown in Figure 7.4 (a) and (b).

2. Full synchronization on the UTA which implements minimum computation on the set of upper event arrival curves (cf. Eq. 7.2) is enforced by the event emitting PTA as depicted in part (d) of Figure 7.4.[1] Full synchronization means that all UTA have to execute their *event*-labelled edge in order to allow the event-emitter (PTA of Figure 7.4 (d)) to release an event `inEvent!`. If not all UTA can participate in the synchronization event emission is blocked.

3. Maximum building on lower staircase curves (cf. Eq. 7.2) is realized by enforcing event emission whenever one of the involved LTA has reached its local threshold N_j, which is achieved by defining the respective invariant for each LTA. A set of LTA and UTA cooperating via the event emitting PTA implements an input generator $\mathcal{G}$ which produces streams w. r. t. a dedicated event type, e. g. of type $inEvent$. Generator $\mathcal{G}(\alpha)$ allows to produce all streams the cumulative bounding functions of which are bounded in the sense of Eq. 7.1 (the proof is provided in [35]). The events emitted by generator $\mathcal{G}(\alpha)$ can now be employed for triggering subsequent behaviour in any downstream TA, e. g. the scheduler model.

Example 7.1. *One may refer to Figure 7.1 in combination with Figure 7.4: for translating upper curve α^{up} one need 2 UTA instantiated with $BMAX := N^{up}_{\{1,2\}}$ and $Delta := \delta^{up}_{\{1,2\}}$. For the event generator*

[1]The location marked with **c** is a committed location, i. e. time is not allow to pass while the PTA resides in this location. A committed location has to be left with the next transition.

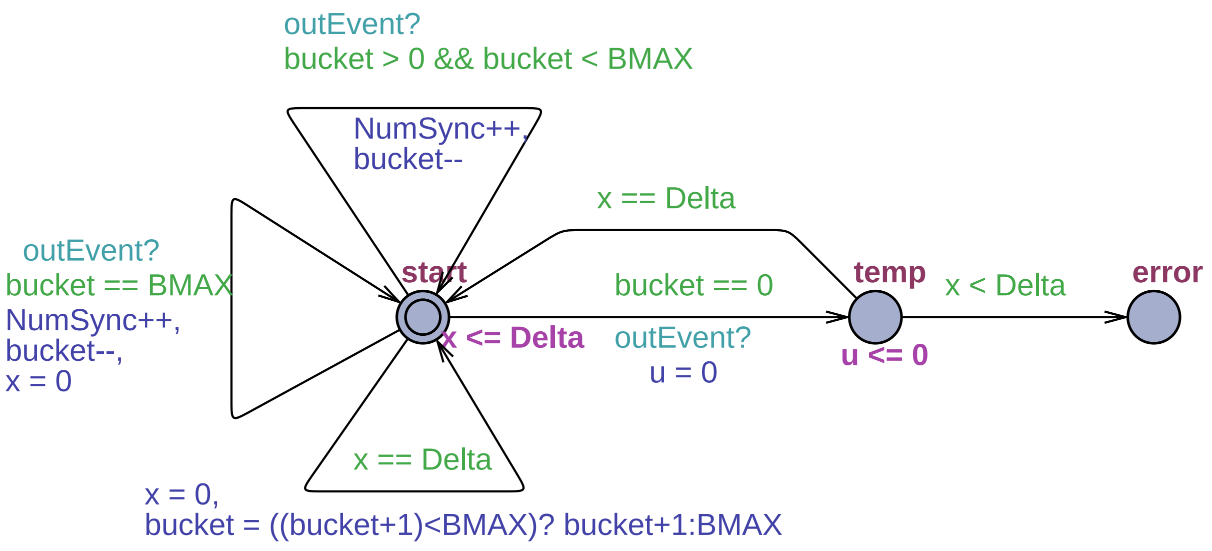
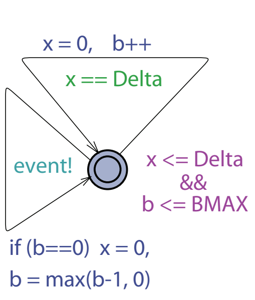
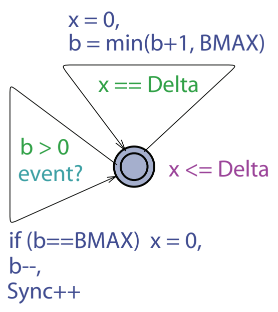


Figure 7.3: Observer UTA



(a) LTA, implements lower curve α_j



(b) UTA, implements upper curve α_i

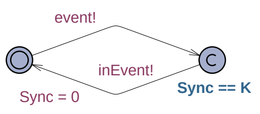
System Declarations

`broadcast channel` *event*;
`int` $BufferSz = 0$;

Local Declarations for LTA, UTA

`const int` BMAX $= |N_{\{i,j\}}|$;
`clock` x;
`int` $b = 0$;
`const int` Delta $= \delta_{\{i,j\}}$

(c) Declarations



(d) TA for enforcing full synchronization

Figure 7.4: TA-based implementation of RTC-based curves

(Figure 7.4 (d)) constant K is set to 2. For implementing the lower curve only a single LTA is needed.

7.3.2 Generalized scheduler model

A simplified generalized scheduler model required by the Quinq tool is presented in Figure 7.5. Let us first discuss only a part (*idle* and *busy* locations) of the generalized scheduler model. The main idea behind this model is that we may represent any resource as a two locations TA. In one of its location this TA is *idle*, i.e., available for the tasks, and in another state it is *busy*, locked for all tasks or tasks with lower priority for non-preemptive or preemptive priority-based scheduling policies respectively. Thus this TA is idle in its initial location until a first event (*inEvent?*) from the event generator described above arrives. Then it switches to the *busy* location resetting on the transition the *clock* used to account for the time passed in the *busy* location. Since other events may arrive while the resource is busy executing the current task, the total remaining execution time will be recalculated and/or events counter incremented. The end of each task execution is marked with an output event broadcast to the system through the

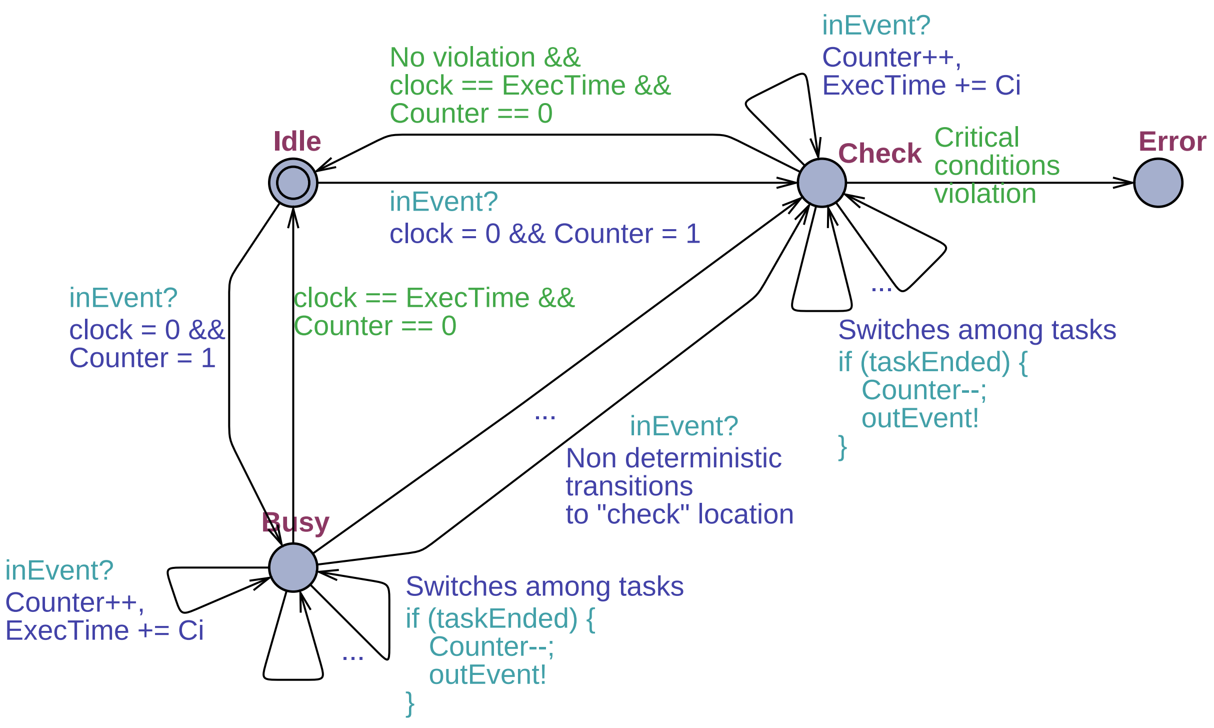


Figure 7.5: Generalized Scheduler Model

outEvent channel. These events are then observed by observers TA (see Section **??**). The scheduler can not go back to the *idle* location unless no more events are stored in the buffer and the system clock is equal to the total execution time. The self-transitions of the *busy* location (task switching conditions) are specific for a chosen scheduling policy. However, for the schedulability analysis the two-location *idle-busy* model should be extended with two more locations: *check* and *error*. Several non-deterministic transitions from the *idle* and *busy* locations to the *check* location should be defined. In the *check* state, the model functions similarly to when it is in the *busy* location (i.e., reacting to the arrival of new events, accounting for the remaining execution time and switching among tasks) plus it checks the violation of critical conditions (e.g., task deadline violation). When the condition of the critical cases is satisfied, the transition to the *error* location will be taken. The example of an extended model of the scheduler with the load-dependent frequency adaptation is presented in Figure 7.5.

7.4 Example case

The TA elements of the case study are presented in Figure 7.6. Given a system with 1 task where the input pattern is a periodic with jitter curves having the UTA linear pattern presented in Figure 7.4 The scheduler of the example load-dependent frequency adaptation scheduler is adapted from [35]. We then extended it with an error state. As for the observer we use only the UTA observer presented in Figure 7.3. The parameters for the simulation are as follows. The scheduler can change its processing speed when the number of tasks in the buffer has exceeded threshold_N = 4. Upon the change in the server speed, the computation time of task changes from C1 = 4 to C2 = 2. The maximum latency that can be tolerated for each task activated is MaxLatency = 20, and the maximum buffer size for the tasks being queued is MaxBufferSize = 10. We also fix the values of BMAXOutUTA = 5, and DeltaOutUTA = 2, respectively

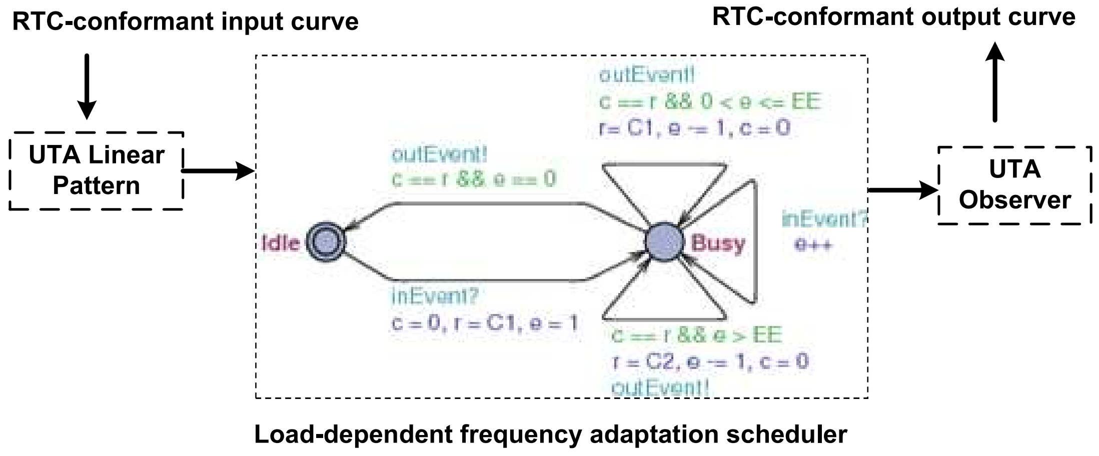


Figure 7.6: Analysed System

the maximum number of tasks coming in burst and the widths of output curve stairs. We would like to find a schedulability region for the BMAXOne, the maximum burst number of activated tasks, and DeltaOne, the widths of input curve stairs, as parameters within the range [0...10]. The result that we obtained is depicted in Figure 7.7.

7.5 Discussion

The work we presented in this chapter conveys the idea of a hybrid design and analysis methodology for distributed real-time systems. The proposed approach couples the Modular Performance Analysis (MPA-RTC) with sensitivity analysis offered in Quinq. It uses a simplified representation of arrival curves to interface heterogeneous modelling components. More specifically, the method automatically converts arrival curves as used by MPA-RTC to Timed Automata models, and uses these models to trigger a state-based and parameterized model of a processing or communication component. In a similar fashion, the output of the component is characterized by appropriate observer automata and automatically converted to arrival curves. The novelty of this work consists in deriving feasible regions for various component parameters such as tolerable data rates or bursts for the input or the output of the component, and tolerable fill levels for its activation buffer. Our results extend previous analysis methods which permitted the evaluation of single design points only. For automatically deriving the region of feasible parameters for a component, we implemented a dedicated tool-chain which employs UPPAAL and NuSMV. The resulting tool permits us to efficiently explore large design spaces and hence directly supports the design of complex distributed systems.

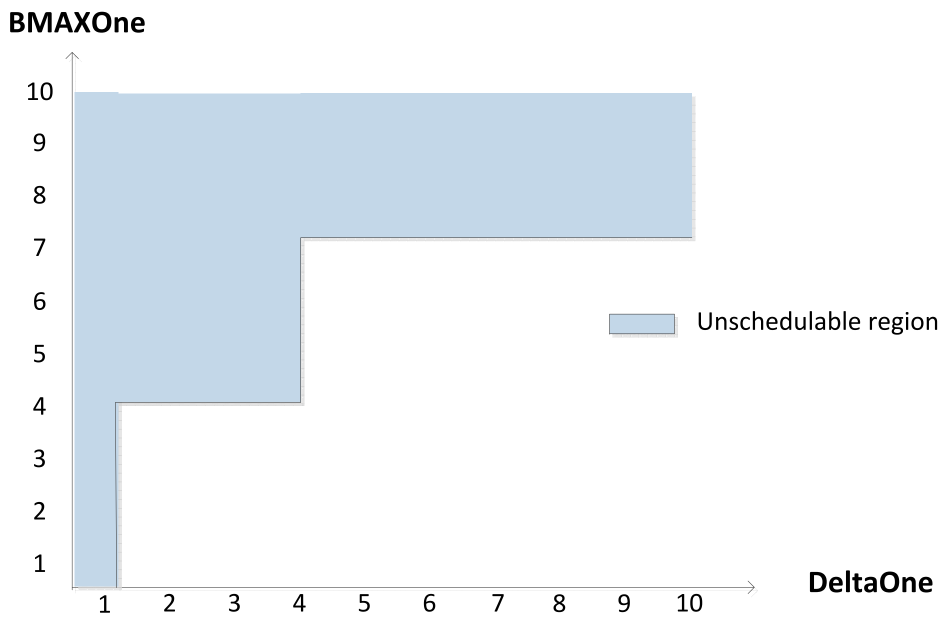


Figure 7.7: Schedulability region

Chapter 8

State of the Art

This chapter reviews the state-of-art related to schedulability analysis using PTA. We started with the overview of previous works on schedulability analysis, continued by works on sensitivity analysis for system schedulability. Last, we offer some comparison to some tools available that are related to sensitivity analysis work.

8.1 Schedulability analysis

A good part of the research on real-time systems has focused on mathematical conditions, whereby a periodic task set scheduled with a fixed priority is schedulable. In their seminal work [42], Liu and Layland identified Rate Monotonic (RM) as the optimal priority assignment to achieve schedulability, and for this assignment they discovered a notorious *sufficient* schedulability condition on the total CPU utilization of the tasks. Albeit very important, this test fails to provide a conclusive response on the system schedulability in many cases of practical and theoretical interest.

Pandya and Joseph [34] proposed an alternative approach. Their methodology allows one to compute the response time of each task, and hence conclude schedulability by comparing it with the deadline. The authors still assume periodic activations for the tasks, but they do not pose restrictions on the priority assignment and on the position of the deadlines. The methodology produces the exact response time if the tasks do not have an initial offset and an overapproximation if the tasks have offsets. Extensions exist for periodic tasks with offsets [46] but they are either intractable or produce conservative results. This analysis techniques have been generalised in different directions. For instance, using the results in [49], it is possible to consider tasks sharing resources.

Our approach differs from the ones in the papers cited above since the proposed SATA technique performs an exhaustive analysis for much more general activation patterns *without* making conservative approximations. This technique lies in the track opened by Wang Yi and his co-workers, who reformulate the problem of schedulability analysis as one of reachability for a network of timed automata. The first attempt in this direction can be found in [45], where the authors use the technology in the UPPAL tool [4] to analyse the schedulability of a set of tasks assuming a non-preemptive policy. This approach of solving the schedulability analysis through reachability problem is also used in [30].

The transition from non-preemptive to preemptive scheduling was not obvious, since many authors used a class of automata for modelling the system (the stop watch automata) [18, 25, 44], for which the reachability problem is known to be undecidable. In [29], the authors proposed a different model for the system based on an extension of timed automata, for which the reachability problem is decidable. In [28], the authors identify an efficient encoding for the schedulability problem in the special but important case of a fixed priority scheduler. In our paper, we generalised this model allowing for parametric constraints associated to the guards and to the invariant conditions of the timed automata.

Parametric timed automata can be found in some previous works. In [55], the authors present a method to model check a real–time system using parametric timed automata when a constraint over the parameter is *given*. This problem is referred to as the "emptiness" problem for PTA and has been prove undecidable, in the general case, in [3]. Interestingly, in [33] a subclass of PTA called L/U PTA is identified for which the problem is decidable, and a solution strategy based on Parametric Difference Bound Matrices is proposed. The solution to the emptiness problem is actually used in this paper as a step to solve a more general problem: *producing* the region of parameters for which the system is unschedulable (namely, the subroutine named `PTA.reachable(Error)` in Figure 4.1). We have provided theoretical evidence that for periodic task sets the algorithm terminates (and hence the problem is decidable). Further investigation on more general examples, along the lines shown in [33], is reserved for future work. This work is extended in [20] by proposing Extended Difference Bound Matrices (EBDM). We will discuss their work in the following section.

In another line of work [6], André shows how given a certain known parameter valuation, one can compute the relaxed constraint on the parameter space that will have the same trace. This approach depends on the input, is not maximal and not guaranteed to terminate. We discuss their tool IMITATOR in the following section.

8.2 Sensitivity Analysis

8.2.1 Analytical approach

The problem of finding the feasibility region was attacked in analytic terms in [11]. The authors considered periodic task sets and improve a methodology, first introduced by Lehoczky and Sha [40], to identify the minimal representation for the region of feasible computation times. This approach is applicable to strictly periodic task sets with zero offsets. As recalled above, we consider much more general activation patterns, although, as a first step, we offer strong convergence results only for the case of periodic task sets with free offsets.

The problem modelling used in the paper is a translation of one explained in Section 2, which considers a task set $\mathcal{S} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i = (C_i, T_i, D_i) is characterized by an activation period T_i, a relative deadline D_i, and a worst case execution time C_i. The activation pattern is assumed periodic with period T_i and offset equal to 0.

The technique is based on the following observation: the worst case situation for the job scheduling is the one in which all tasks start at the same time (which happens at time 0). In other word if the job starting at zero is schedulable so are the other jobs. The idea of time demand approach is the following: a task is schedulable if in the time interval $[0, D_i]$ there exists a time t in which the computation demand $L_i(0,t)$ of task τ_i and of higher priority tasks and of task τ_i is lower than the computing time offered by the CPU (e.g., t):

$$\exists t \text{ such that } L_i(0,t) \leq T \rightarrow \tau_i \text{ schedulable}$$

Luckily it can be shown that the verification can be done only on a limited set of points:
$G_i = \left\{ rT_j : j = 1, \ldots, i, r = 1, \ldots, \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\}$.

Therefore the task is schedulable if:

$$L_i = \min_{t \in G_i} \frac{L_i(0,t)}{t} \leq 1 = \min_{t \in G_i} \frac{\sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \leq 1$$

.

The main contribution of the paper is the computation of a small set of points instead of S_j that preserve schedulability leading to the following:

Theorem 8.1. When deadlines are equal to periods, the region of the schedulable task sets R defined as

$$R(T_1, \ldots, T_n, D_1, \ldots, D_n)|_{D_i = T_i} = \{(C_1, \ldots, C_n) \in \mathcal{R}_+^n : S \text{ is schedulable}$$

is given by :

$$R(T_1, \ldots, T_n, D_1, \ldots, D_n)|_{D_i = T_i} = \{(C_1, \ldots, C_n) \in \mathcal{R}_+^n :$$

$$\bigwedge_{i=1\ldots n} \bigvee_{t \in G_i} \sum_{j=1}^{i} \lceil \frac{t}{T_j} \rceil C_j \leq t\},$$

$$\text{where} G_i = \{rT_j : j = 1 \ldots i, r = 1 \ldots \lfloor \frac{T_i}{T_j} \rfloor\}$$

Not only does the theorem above allow us to decide the schedulability of a task set but it also allows us to compute the entire feasibility region. As we have shown in our work, our approach allows us to find the same result for a much larger class of task sets.

8.2.2 Sensitivity Analysis Tools

In this section we discuss several other sensitivity analysis tool that are directly related to our work. We discuss an overview of their approach then provide some discussion on the comparison of their approach and ours.

8.2.3 Modeling and Analysis Suite for Real-Time Applications (MAST)

MAST[1] is an open source tool developed to model real–time system that facilitates some scheduling analysis on the model: worst-case response time schedulability analysis, blocking time analysis, optimum priority assignment analysis, and most relevant to out subject, sensitivity analysis [32]. A real-time situation is modelled as a set of concurrent transactions that compete for the resources offered by the platform. Tasks can be periodic or aperiodic. Periodic tasks have the variables of period, deadline, and computation time, while the aperiodic tasks have the variables of minimum inter-arrival time, computation time, and deadline after each arrival. Unlike in our case, they do not have offset as possible variables in their periodic tasks. However they employ the other notion of *offset* in their analysis of WCRT. The authors use this notion of *offset* to account for the dependency and precedence relation between jobs in their analysis. While in our case an offset is a delay with which we offset the first activation of our jobs in a task, in MAST, an offset is a delay with which we offset all the activation of our jobs in a task.

On the real–time system modelled in their tool, they provide sensitivity analysis through the calculation of slacks in their system and transaction. *Slacks* are the percentages by which the execution times

[1]http://mast.unican.es/

of the operations of a transaction (or of the system) can be increased while keeping the system schedulable (or decreased to make it schedulable if it was not schedulable.). There are different kinds of slacks provided:

- System slack: affects all the operations in the system
- Processing resource slack: affects only the operations executed in a given processing resource
- Transaction slack: affects only the operations used in a given transaction
- Operation slack; affects only one single operation

The slacks are calculated by modifying the worst-case execution times and repeating the analysis using a binary search to find the point in which the system becomes unschedulable or schedulable. The slack is calculated with a 1% precision to limit the amount of times the analysis is repeated to around 20 times.

In term of real–time system modelling, compared to our tool, MAST already is already matured as it provides support for several scheduling policies in system with single as well as multiprocessor. This tool also supports the shared resources in between the tasks and arranges the subsequent blocking and mutual exclusion use of the shared resources. However, while the sensitivity analysis information of the system slack is valuable, the fact that the tool only calculate the slack in the term of percentage does not provide the exact value information to the user. Each of the feedback is highly related to the reference valuation given as input. Also, the slacks are end-to-end slack: Transaction Slacks, System Slacks, Processing Resource Slacks, and Operation Slacks. While these provide information to some level, they do not directly relate to specific variables.

8.2.4 SMART

SMART is the tool built as a proof of concept for a work on parametric timed automata analysis using extended difference bounded matrix (EBDM) presented in [20]. This tool is not available online, therefore we will discuss it only based on the PhD thesis of the author. EDBM approach is developed to improve the Parametric difference bounded matrix approach presented in [8] that is then implemented in the tool TReX [7]. PDBM extends the classical BDM in allowing parameters in the arithmetical constraints defined as the parametric bound for difference among two clocks. In this work, symbolic reachability analysis of the parametric system is performed through the operation on PDBM the same way we perform operations on DBM: through intersection, time elapse, and clock reset. However, parameters are involved as the bounds could contain parameters.

EDBM differs to PDBM by having parameters represented in the matrix the same way the clocks are represented. Thus the entries of the EDBM, i.e. all the bounds stored in the structure, are all numerical. Using this approach the operations that requires comparing two elements of the matrix are straightforward. All expressions that are difference of two sums of clocks may be handled be EDBM. As result, the size of the matrix is constant during entire analysis. This is its edge over PDBM in which parameters are constrained separately from the main data structure which only constraints allowed values on clocks.

The symbolic reachability analysis of the system represented in SMART is done by performing symbolic forward and analysis that performs intersection, reset, backward and forward time elapse operation on the EBDM. These operations will result in a polyhedra in the space of clocks and parameters. In the perspective of the result, the poyhedra result obtained by performing those operations on EBDM can be compared with the polyhedra obtained by our tool. Provided SMART could find all the possible traces to the error location, then given a system that could be accepted in SMART, theoretically the result provided by Quinq on the system would be the same with the polyhedra represented through EBDM. However that is not the case in the other way around, as Quinq could work on system with more general constraints that might not be acceptable for EDBM requirement.

The difference of the modelling provided in SMART to ours are the following :

1. The clock and variable updates

   SMART tool only allows the clocks to be reset to zero. Variables, in the other hand, can only be updated with another value resulting from any algebraic expression. It may not contain any parameter nor clocks. In our case, any linear constraints involving parameters, clock, other variables, and constant are allowed to be used as arbitrary clock and variables updates.

2. Constraints on invariants and guards

   SMART allows two possible forms of expressions : 1) Difference of two clocks, parameters, and real numbers. 2) Any algebraic expression defined with variables and reals. These constraints in expression forms do not apply to our modelling approach. As in our update statements, we allow any linear expression made from parameters, clocks, other variables, and reals as the invariants and guards on our model.

In term of the analysis method, our tool differs to SMART in the generality of our method towards any possible expressions. In SMART, it is required that the form of all expressions that are used in transition guards or invariants of the analysed automata must be known a priori to the analysis. As it is needed to formulate the matrix columns and rows. Before the analysis may start, all the model must be scanned in order to find out what expressions are used in it.

And last, as other approaches based on DBM, it is necessary to calculate the closure of the EDBM used for the system analysis. The cost of calculating closure depends exponentially on the number of clocks and parameters and also on the complexity of used expressions. This complexity is expressed in a size of sets labelling rows and columns of EDBM, corresponding to number of clocks and parameters in expression guarding transitions of the automata. This is not the case for our method approach. We do not need to calculate the closure for each clocks and parameters involved in our system to converge in our method.

8.2.5 Imitator

Given Linear Hybrid Automata problem with parameters, certain intended behaviour, and a rectangular parameter space IMITATOR tool employs an algorithm called inverse method [6] to define the largest set of parameters where the sytem will behave as intended.

The inverse method is done as follows: given one set of parameters valuation, from the initial state, using quantifier elimination on the clocks, invariant and guard of states to obtain each reachable states constraint, collect all the negated constraints of states that do not comply with the input parameter valuation. Then we recompute from initial states all the updated reachable states with each collected negated constraints until we have all states that are reachable by the input points and the collected constraints. At the end, the intersection of the remaining state constraints projected into parameters space is returned. The result of each inverse method given a valuation of parameter set is a convex set of instantiations of the parameters such that all instantiations will cause the same traces as the traces of the instantiation given in the input.

The authors then extend this inverse method to achieve the mapping of the parametric space into behavioural tiles. They use the term behavioural cartography [5] for this process. The mapping is acquired by enumerating random integer points or a specific step size of points in the space of parameter (thus the number of iteration is finite) then constructing a region / tile around the point where all points have the same behaviour, named behavioural tile, using inverse method. This process is repeated until the whole parameter space targeted is covered by the tiles, Then they categorize the tiles into good and bad zones according to the set of traces that can be achieved by points in each tiles. A tile is ”good” if all of its traces is ”good” , and a tile is ”bad” if at least one of the trace is ”bad”. This bad/good property is decided on the reachability of bad states on the trace. With that, the state space is then divided into good tiles zone and bad tiles zone, and the good tiles zone is returned as the solution for the problem.

The inverse method that they are using relies on having a certain parameter instantiation of the system to further produce the constraints on the parameters that will guarantee the system to have the same behaviour.

This method then can be used in 2 scenarios :

1. Having ”good” parameter instantiation as input to then generalize it, producing an area in parameter space that corresponds with the same good behaviour properties.

2. In the case no property verification is possible, having points in the parameter space as input to then generalize each of them, producing areas corresponding to different possible behaviours of the system.

As it relies on having known a ”good” parameter instantiation as input to be able to come up with constraint for parameters that satisfies the same good property. The method technically is only able to verify that the input instantiation of the parameter and the output constraints of the parameters have the same behaviour. It relies on other system to provide with the input having the knowledge of at least an instance of the parameters that is verified to satisfy certain property. Or in the other case, where cartography feature is used, it then needs other system to then verify whether each block satisfy certain property or not.

The difference of their approach to ours lies in the generality of the problem tackled. It can be said the main problem that they are working on is a sub-unit of our problem. In this light, their notion of *parameter* in our system is different compared to the *parameter* notion in IMITATOR. In Imitator, *parameters* are all involved variables in a system. Given a valuation reference for all the *parameters*, they will try to find all valuation for those which will result in the same behaviour, i.e. traversing the same locations and transitions. In our case *parameters* are a portion of all the variables in a system, which all valuation we will try to find to decide whether the system will be schedulable or not schedulable. The problem that they are working on is, given a reference valuation, they will provide the constraints that will describe the region with the same behaviour in the parameter area. While the problem at our hand is: given a system, we would provide all the constraints describing the schedulability region in the parameter area.

This comparison can be seen clearly in their case studies [2] in which two of the case they are working on are from our previous work in [21] and [39]. The result constraint that they are obtained describes the area that is strictly less and included in our final constraints, because it is the area in which their reference valuation input is valid.

In the case of performing cartographical mapping on the parameter area, where they would iteratively perform the algorithm for all points, they could theoretically end up with our result. However, the integer domain in which they work on would prevent them from capturing areas with non monotonic behaviour in between the integer points. And performance-wise, their algorithm is not suitable as . Replicating

[2]http://www.lsv.ens-cachan.fr/Software/imitator/case-studies.php

our case study in [21] for one valuation point, which result in a sub region in our result, took their time the longest compared to other studies due to the large number of possible states that they ended up numerating for this area.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this dissertation, we are working on the field of real–time system design in the strive for achieving correct and robust system. We acknowledge the importance of proper modelling and design platform to enable correct design of the system. For the robustness of the system, we stress the importance of system robustness evaluation and parameter tuning. We then provide our solution for this goal : Modelling via parametric timed automata (PTA), Parametric Verification of Temporal Properties (PTVP) algorithm, and Quinq tool.

PTA representation is proposed as this approach will enable the modelling of real system with arbitrary activation pattern and scheduling policy. The timing property of the real–time system will be enforced through the constraints in the guard and invariant of the PTA. Our definition of PTA enabled any linear operation on the clocks and system variables as the guard and invariant constraints. The schedulability problem of the system is then transformed into the reachability problem of locations associated with violation to timing properties.

Working on the model in PTA, we presented a general methodology for the symbolic computation of the schedulability region, our PTVP algorithm. The resulting schedulability region will provide the user with the robustness evaluation of the system as well as the information on the possible parameter tuning. The methodology applies to very general activation patterns and it does not make any conservative approximations, and is implemented by integrating traditional symbolic model checking and SMT solvers. We proved convergence in the important case of fixed priority scheduling for periodic task systems. We have also provided a way to complete the method by performing inductive reasoning to the final result.

We have also presented our tool Quinq with which we facilitate the modelling of real–time system through parametric real timed automata. The PTVP algorithm has been implemented using NuSMV3 as blackbox components. The method has been automated from end to end: from the input format where we enable three kinds of possible input format, up to the drawing of the result region. Optimization possibilities for the search method using non-parametric model checker is also offered.

While there has been many development in the area of sensitivity analysis on parametric system in recent years, our tool still has its edge in the field of the goal that we would like to work on: defining schedulability region in parameter space of a real system. Compared to the other tool for sensitivity analysis, the method and tool that we used offer more freedom in parametric definition in the system: any design variables can be a parameter. Unlike the other tools we described, we do not rely on having a reference input first to start the schedulability region analysis. Thus, in term of the result, we provide the largest possible view of the schedulability in the parameter space. Working in rational domain, we provide the exact area for each parameter in which the designer could later decide their optimum choice for parameter value towards the maximum system robustness.

9.2 Future Work

There are several aspects that will still need to be further researched. On the theoretical side, our most important goal is to extend the study of convergence to the general case. Thorough performance analysis of the tool as well as benchmarking study will also help us to better evaluate the tool performance. In term of the real–time system domain, working on case studies involving dependencies between tasks, multi-processor, as well as other scheduling policies will add the portfolio for this method. Another interesting possibility is to use the schedulability region as a design tool by introducing appropriate cost functions.

Bibliography

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. ISSN 0304-3975.

[2] Rajeev Alur and David L. Dill. Automata For Modeling Real-Time Systems. In Mike Paterson, editor, *Proc. of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

[3] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC'93*, pages 592–601, New York, NY, USA, 1993. ACM. ISBN 0-89791-591-7.

[4] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times - a tool for modelling and implementation of embedded systems. In *TACAS'02*, pages 460–464, London, UK, 2002. Springer-Verlag. ISBN 3-540-43419-4.

[5] Étienne André and Laurent Fribourg. Behavioral cartography of timed automata. In Antonín Kučera and Igor Potapov, editors, *Proceedings of the 4th Workshop on Reachability Problems in Computational Models (RP'10)*, volume 6227 of *Lecture Notes in Computer Science*, pages 76–90, Brno, Czech Republic, August 2010. Springer.

[6] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5): 819–836, October 2009.

[7] A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems, 2001.

[8] Aurore Annichini, Eugene Asarin, and Ahmed Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proceedings of the 12th International Conference on Computer Aided Verification*, CAV '00, pages 419–434, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67770-4.

[9] Gilles Audemard, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani. Bounded model checking for timed systems. In *FORTE'02*, pages 243–259, London, UK, 2002. Springer-Verlag. ISBN 3-540-00141-7.

[10] Gerd Behrmann, Re David, and Kim G. Larsen. A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, pages 200–236. Springer, 2004.

[11] Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Comput.*, 53(11):1462–1473, 2004. ISSN 0018-9340.

[12] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *LNCS*. Springer, 2001.

[13] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani. The MathSAT 3 system. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20*, volume 3632 of *Lecture Notes in Artificial Intelligence*, pages 315–321. Springer, 2005.

[14] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *CAV'08*, LNCS, Princeton, USA, 2008. Springer.

[15] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. Delayed theory combination vs. nelson-oppen for satisfiability modulo theories: A comparative analysis. In *LPAR*, pages 527–541, 2006.

[16] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency*. Series: Series in Computer Science. Springer, 2005.

[17] Luca P. Carloni, Fernando De Bernardinis, Alberto L. Sangiovanni-Vincentelli, and Marco Sgroi. Platform-based and derivative design. In *The Industrial Information Technology Handbook*, pages 1–15. 2005.

[18] Franck Cassez and François Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In *CAV'00*, pages 373–388, London, UK, 2000. Springer-Verlag. ISBN 3-540-67770-4.

[19] Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R. K. Shyamasundar. Computing Predicate Abstractions by Integrating BDDs and SMT Solvers. In *FMCAD'07*, pages 69–76, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3023-0.

[20] Patryk Chamuczyński. *Algorithms and data structures for parametric analysis of real time systems*. Doctorate thesis, University of Göttingen, Germany, 2009.

[21] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *Real-Time Systems Symposium, 2008*, pages 80–89, Dec. 3 2008.

[22] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. Nusmv: A new symbolic model verifier. In *CAV'99*, pages 495–499, London, UK, 1999. Springer-Verlag. ISBN 3-540-66202-2.

[23] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In *TACAS'08*, volume 4963 of *LNCS*, pages 397–412. Springer, 2008. ISBN 978-3-540-78799-0.

[24] Edmund Clarke, Daniel Kroening, Joel Ouaknine, and Ofer Strichman. Computational challenges in bounded model checking. *Software Tools for Technology Transfer (STTT)*, 7(2):174–183, April 2005.

[25] James C. Corbett. Modeling and analysis of real-time ada tasking programs. In *RTSS'94*, pages 132–141, 1994.

[26] M. W. Dawande and J. N. Hooker. Inference-based sensitivity analysis for mixed integer/linear programming. *Oper. Res.*, 48(4):623–634, 2000. ISSN 0030-364X.

[27] EADS Innovation Works. Case study on distributed heterogeneous communication systems (HCS), 2009, `http://www.combest.eu/home/?link=Application1`.

[28] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis using two clocks. In *TACAS'03*, volume 2619 of *LNCS*, pages 224–239. Springer, 2003.

[29] Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *TACAS'02*, pages 67–82, London, UK, 2002. Springer-Verlag. ISBN 3-540-43419-4.

[30] A.N. Fredette and Rance Cleaveland. Rtsl: a language for real-time schedulability analysis. In *RTSS'93*, pages 274–283, Washington, DC, USA, 1993. IEEE Computer Society. ISBN 0-8186-4480-X.

[31] Giorgio C. Butazzo. *Hard Real-Time Computing Systems*. Springer, 2005. ISBN 0-387-23137-4.

[32] M. Gonzlez Harbour, J. J. Gutirrez Garca, J. C. Palencia Gutirrez, and J. M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *In 13th Euromicro Conference on Real-Time Systems*, page 125, 2001.

[33] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. In *TACAS'01*, pages 189–203, London, UK, 2001. Springer-Verlag. ISBN 3-540-41865-2.

[34] Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[35] Kai Lampka, Simon Perathoner, and Lothar Thiele. Analytic real-time analysis and timed automata: A hybrid method for analyzing embedded real-time systems. In *EMSOFT '09: Proceedings of the 7th ACM international conference on Embedded software*, pages 107–116, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-627-4.

[36] Kai Lampka, Simon Perathoner, and Lothar Thiele. Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems. *Design Automation for Embedded Systems*, 14(3):193–227, 2010.

[37] K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[38] Hoa Thi Thieu Le, Luigi Palopoli, Roberto Passerone, and Yusi Ramadian. Modeling a distributed heterogeneous communication system using parametric timed automata. Technical Report DISI-10-031, Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, April 2010.

[39] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, Yusi Ramadian, and Alessandro Cimatti. Parametric analysis of distributed firm real-time systems: A case study. In *ETFA*, pages 1–8, 2010.

[40] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *RTSS'89*, pages 166–172, 1989.

[41] Joseph Y.-T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Inf. Process. Lett.*, 11(3):115–118, 1980.

[42] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973. ISSN 0004-5411.

[43] A. Marchand and M. Silly-Chetto. Qos scheduling components based on firm real-time requirements. In *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, page 141, 2005.

[44] Jennifer McManis and Pravin Varaiya. Suspension automata: A decidable class of hybrid automata. In *CAV'94*, pages 105–117, London, UK, 1994. Springer-Verlag. ISBN 3-540-58179-0.

[45] Christer Norström, Anders Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *RTCSA'99*, page 182, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0306-3.

[46] J. C. Palencia and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS'98*, page 26, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-9212-X.

[47] PTP. A precision clock synchronization protocol for networked measurement and control systems. Ieee standard 1588-2002, November 2002.

[48] A. Sangiovanni-Vincentelli. Defining platform-based design. 2002.

[49] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, 39:1175–1185, September 1990. ISSN 0018-9340.

[50] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In *FMCAD'00*, pages 108–125, London, UK, 2000. Springer-Verlag. ISBN 3-540-41219-0.

[51] Alena Simalatsar, Yusi Ramadian, Kai Lampka, Simon Perathoner, Roberto Passerone, and Lothar Thiele. Enabling parametric feasibility analysis in real-time calculus driven performance evaluation. In *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*, CASES '11, pages 155–164, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0713-0.

[52] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. Intl. Symposium on Circuits and Systems*, volume 4, pages 101–104, 2000.

[53] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System Architecture Evaluation Using Modular Performance Analysis: A Case Study. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):649–667, 2006.

[54] Farn Wang. Formal verification of timed systems: A survey and perspective. *IEEE*, 92, August 2004.

[55] Dezhuang Zhang and Rance Cleaveland. Fast on-the-fly parametric real-time model checking. In *RTSS'05*, pages 157–166, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2490-7.

Appendix A

Quinq Usage

A.1 Sensitivity add-on

A.1.1 Input

As an add-on in NuSMV3 the functions in sensitivity add-ons are available via NuSMV3. The input file is read by the following command :

```
esmc > read_model -i input.smv
```

esmc is the NuSMV3 executable.

Or we can input them as we call the NuSMV itself:

```
./esmc -int -cpp input.smv
```

The option −int is to enter the interactive mode of NuSMV3, and the option −cpp is to perform c pre-processing to the input file.

A.1.2 Functions

We provide the following functionalities in sensitivity add-on :

Schedulability analysis

This main function is implemented in the function sensitivity_analysis. We call this function as follows:

```
esmc > sensitivity_analysis
```

followed by the desired options. Some features that are enabled in the options are:

- The recursion depth

  This setting will allow us to set more recursion depth on cases with a lot of tasks involved.

  ```
  -depth max_depth
  ```

  Maximum recursion depth.

- The BMC bound

  ```
  -bound max_k
  ```

  With this option, Quinq will continuously do the iteration up to length max_k is reached. The default is checking up to length 30.

- Verbose possibility

  This option will enable print out of the system phases that have been performed.

  ```
  -v <verbose_level>
  ```

  On $verbose\_level \geq 1$ Quinq will dump every traces found in XML format.

- Quantifier method As we provide two different methods in performing quantifier elimination in the constraint processing, user may choose between the two methods using this option:

  ```
  -quant quantifier
  ```

  Using projection method we input the value 0 while using Fourier Motzkin quantifier we specify the value 1.

- printing trace options Aside from using the verbose, we can also specify to request the example trace of the model to be printed with the following option:

  ```
  -print_trace
  ```

- performing inductive reasoning Using this option Quinq will perform inductive reasoning with interpolants at the last known length in the end of the iteration. The option is :

  ```
  -inductive
  ```

- performing single iteration or all possible iteration up to the bound

  For testing purposes we might want to perform only one iteration in searching for the constraint. We use the following option for this :

  ```
   -single
  ```

- parameter file

  By using this option we can provide an input file enlisting all the parameters in the model.

  ```
  -param parameter_file
  ```

  Input file $parameter\_file$ has the format of:

```
<param1 name>
...
<param_n name>
```

- Continuous variables Using this option we can specify a text input file enlisting all the continuous variables in the model.

  ```
  -cont continuous_vars_file
  ```

- Index of checked property Checks the INVAR property specified with index idx. The default is checking property 0. This option is used when we have more than one property to be checked in our model.

  ```
  -propnbr idx
  ```

Schedulability analysis by trace

This function is titled trace_sensitivity_analysis. To call a schedulability analysis on the model based on trace given in trace.xml, which is a NuSMV trace in the XML format, we call the following command:

```
esmc > trace_sensitivity_analysis trace.xml
```

followed by the appropriate options :

- parameter file

  By using this option we can provide an input file enlisting all the parameters in the model.

  ```
  -param parameter_file
  ```

- Continuous variables Using this option we can specify a text input file enlisting all the continuous variables in the model.

  ```
  -cont continuous_vars_file
  ```

- Verbose possibility

  This option will enable print out of the system phases that have been performed.

  ```
  -v <verbose_level>
  ```

  On $verbose\_level \geq 1$ Quinq will dump every traces found in XML format.

- Quantifier method As we provide two different methods in performing quantifier elimination in the constraint processing, user may choose between the two methods using this option:

  ```
  -quant quantifier
  ```

  Using projection method we input the value 0 while using Fourier Motzkin quantifier we specify the value 1.

- printing trace options Aside from using the verbose, we can also specify to request the example trace of the model to be printed with the following option:

  ```
  -print_trace
  ```

- The recursion depth

  This setting will allow us to set more recursion depth on cases with a lot of tasks involved.

  ```
  -depth max_depth
  ```

  Maximum recursion depth.

Point checking

This is the function to check whether a point is already covered in the already found schedulability region. The input file for this function is an smv file with the parameter variables assigned to the value of the point that we want to check and the current unschedulable region constraints. After registering the input file, we then call the following command:

```
esmc > sensitivity_check_param -i index
```

where index is the index of the INVARSPEC where we define the inclusion specification in the model.

A.2 Periodic real–time system automatic schedulability analysis

A.2.1 Input model

We provide an automatic way to generate a NuSMV model of periodic case with periodic task activation pattern and scheduling policy of preemptive, fixed priority scheduler. We also produce as a by-product an XML model for this standard periodic case. The XML file is viewable in UPPAAL as UPPAAL input file for graphical purpose. The user only needs to specify the following for the automatic model generator:

1. Number of tasks involved

   We specify the number of tasks involved with the variable N. For example, on a system with 5 tasks we declare them as follows :

   ```
   const int N = 5;
   ```

2. Fixed task parameter specification

   For each of the N tasks specified, we need to specify the period T, $computationtime$c,deadline D, and offset $offset$ in an array ordered by their priority number. The first entries in each of the array describes the parameters for the task with the first priority in the system. For example we may describe the system with 5 tasks as follows:

   ```
   int D[N] := {20, 30, 45, 55, 60};
   int C[N]:= {1, 1, 2, 2, 3};
   int T[N]:= {20, 30, 45, 55, 60};
   int offset[N]:= {2, 2, 2, 2, 2};
   ```

The above notation means we have $S = \{\tau_1, \tau_2, \ldots, \tau_5\}$ where, the deadlines and periods of each tasks are $D_1 = T_1 = 20$, $D_2 = T_2 = 30$, $D_3 = T_3 = 45$, $D_4 = T_4 = 55$, and $D_5 = T_5 = 60$, and their computation times $C_1 = 1$, $C_2 = 1$, $C_3 = 2$, $C_4 = 2$, and $C_5 = 3$. The offsets for each of the tasks are $O_1 = 2$, $O_2 = 1$, $O_3 = 2$, $O_4 = 2$, and $O_5 = 2$.

3. Parameter and their value range

   The parameter that needed to be analysed is encoded with -1 value in the array. For example if we want to define the computation time of the second task as the parameter, we will write the following:

   ```
   int C[N]:= {1, -1, 2, 2, 3};
   ```

   For each of the parameter we decide, we need to define their value ranges for the search to converge. The value ranges are defined in additional parameter with name in the format $variable\_lower$ for the lower boundary of the value range and $variable\_upper$ for the upper boundary on the value range. For example, defining that the lower boundary for the parameter computation time above is encoded as follows :

   ```
   int C_lower[N]:= {0, 0, 0, 0, 0};
   ```

   The encoding describes that the lower boundary for all of the tasks computation time are zero. However, as we only declared C_2 as parameter by assigning value -1 to it, only the second entry in the lower boundary will be considered.

4. Instantiation of the processes

   We need to instantiate all the N processes that we declared

   ```
   process_1 = generic_process(0);
   process_2 = generic_process(1);
   process_3 = generic_process(2);
   process_4 = generic_process(3);
   process_5 = generic_process(4);
   scheduler = generic_scheduler(4);
   ```

   And enlist one or more processes to be composed into a system.

   ```
   system process_1, process_2, process_3, process_4, process_5, scheduler;
   ```

It is sufficient using the templates and the above encoding as input for Quinq.

A.2.2 Schedulability analysis for periodic case

We use the option :

```
-periodic
```

To perform sensitivity analysis on periodic cases with preemptive, fixed priority scheduling policy With periodic cases using the we need to specify the following input :

1. How many tasks we would like to analyse their schedulability? This is done by specifiying the minimum and maximum id of task that we would like to check

   ```
   -maxtask max_t
   ```

   Executing sensitivity analysis up to the max_t-th task.

   ```
   -mintask min_t
   ```

   Executing sensitivity analysis from the min_t-th task.

2. The granularity of the search space
3. The xml file name

A.3 UPPAAL - NuSMV plugin

A.3.1 UPPAAL model for template for NuSMV

we enable a plugin to automatically translate UPPAAL model into NuSMV. The path to convert is the following :

- Translate XML to SMV using JUNT
- Check the correctness of the translation:
  - Parameter naming (from array to non -array)
  - check the specification
  - check the temporary last step additional invar
  - check the parameters are in frozenvar state (and no assignment to it)
  - add the initial constraints for the parameters :

    ```
    INVAR param1 >= ZERO
    INVAR param1 <= upper_bound
    ...
    INVAR param_n >= ZERO
    INVAR param_n <= upper_bound
    ```

- Create the uppaal query in accordace, modify verifyta.sh
- prepare the parameter specification file

  ```
  <nbr of parameter>
  <param1 name> <lower bound> <upper bound>
  ...
  <param_n name> <lower bound> <upper bound>
  ```

- Modify the script with cleaning up purposes

A.3.2 Schedulability analysis with optimized search

With periodic cases using the we need to specify the following input :

1. Which mode we would like to use :

   We provide several possibilities to do the analyzation

   - Only in UPPAAL Executing sensitivity analysis from UPPAAL traces The keyword :

     ```
     -withoutNuSMV
     ```

   - Only in NuSMV
     Only executing sensitivity analysis using purely NuSMV.
     The keyword :

     ```
     -withoutUPPAAL
     ```

   - With both UPPAAL and NuSMV This is the default approach

2. The search optimization possibility

   First, as we presented above that there are several ways to generate the points for search optimization, we can choose from the following:

   - Datagrid search
     This is the default option.
   - Random search
     Executing sensitivity analysis from UPPAAL traces on randomly generated points.

     ```
     -randomSearch
     ```

   - Model-checker generated points
     Executing sensitivity analysis from UPPAAL traces on points generated by NuSMV.

     ```
     -automaticPtGen
     ```

     This approach does not work very well with NuSMV and UPPAAL so far though as we could not really ask them to enumerate all the possible points / error traces.

   Aside from that we need to specify the granularity of the search space :

   ```
   -datapoint datapt_nbr
   ```

   This will provide us with preliminary search on *datapt_nbr* number of data in each parameter area. The default number is 3. And if we choose to do the search from points with big values first up to points with smaller values, we can enable this option:

   ```
     -inverseSearch
   ```

   To perform search on gridspace using NuSMV from the biggest points up to the smallest.

   After obtaining the points from the desired generation methods we have the following possibility of performing the search:

- Using UPPAAL to do the search This is the default option in performing the search. This is chosen as it is faster, relying on numerical, explicit timed automata representation approach. However the granularity in this approach is limited by the fact that UPPAAL works on integer. Thus the smallest granularity that we can check are values of 1.0.
- Using NuSMV to do the search
  To perform search on gridspace using NuSMV.

  ```
  -nusmvSearch
  ```

  This approach provides the possibility to search up to the smallest granularity possible. However using symbolic model checking, the approach could not be as fast as UPPAAL.

3. The xml file name The input is an xml file that will be a guide

   ```
   -withoutUPPAAL
   ```

4. The parameter file

   ```
   -pspec spec_file
   ```

   Specification file for general case of sensitivity analysis.

5. BMC bound

   ```
   -boundTrace
   ```

   Maximum bound for search on gridspace using NuSMV.

6. Optimization of bound

   ```
   -k_addition k_add
   ```

   Search up to k_add steps after the last k acquired from the last trace.

7. Performing trace verification on an instant point

   ```
   -verifytrace xml_file
   ```

   XML file with instantiation on parameters on which the trace will be verified.

8. Verbose option

   ```
   -v verbose_level
   ```

   Printing out program debugging messages at level $verbose\_level$.

A.4 Features

A.4.1 Drawing plugin

Example command :

```
./drawme result.exp1 300
```

This command will draw the result constraints provided in the file `r`esult.exp1 using the resolution of 300.