Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Biblioteca Digital FCEN-UBA

Modelando y verificando diseños de sistemas de tiempo real

Braberman, Víctor Adrián

2000

Tesis Doctoral

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

www.digital.bl.fcen.uba.ar

Contacto: digital@bl.fcen.uba.ar

Este documento forma parte de la colección de tesis doctorales de la Biblioteca Central Dr. Luis Federico Leloir. Su utilización debe ser acompañada por la cita bibliográfica con reconocimiento de la fuente.

This document is part of the doctoral theses collection of the Central Library Dr. Luis Federico Leloir. It should be used accompanied by the corresponding citation acknowledging the source.

Fuente / source:
Biblioteca Digital de la Facultad de Ciencias Exactas y Naturales - Universidad de Buenos Aires

BIBLIOTECA CENTRAL
FACULTAD DE CIENCIAS EXACTAS
Y NATURALES / UBA
FCE Y N BIBLIOTECA

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Modeling and Checking Real-Time System Designs

by

Víctor Adrián Braberman

Director: Ph.D. Miguel Felder

Pabellón 1 - Planta Baja - Ciudad Universitaria
(1428) Buenos Aires
Argentina

e-mail:vbraber@dc.uba.ar
http://www.dc.uba.ar/people/exclusivos/vbraber

3287

BIBLIOTECA CENTRAL
FACULTAD DE CIENCIAS EXACTAS
Y NATURALES / UBA

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Modelando y Verificando Diseños de Sistemas de Tiempo Real

Autor:

Víctor Adrián Braberman

Director: Doctor Miguel Felder

Pabellón 1 - Planta Baja - Ciudad Universitaria
(1428) Buenos Aires
Argentina

e-mail:vbraber@dc.uba.ar
http://www.dc.uba.ar/people/exclusivos/vbraber

82274

Nº 32879

E2

Abstract

Real-time systems are found in an increasing variety of application fields. Usually, they are embedded systems controlling devices that may risk lives or damage properties: they are safety critical systems. Hard Real-Time requirements (late means wrong) make the development of such kind of systems a formidable and daunting task. The need to predict temporal behavior of critical real-time systems has encouraged the development of an useful collection of models, results and tools for analyzing schedulability of applications (e.g., [109]). However, there is no general analytical support for verifying other kind of high level timing requirements on complex software architectures. On the other hand, the verification of specifications and designs of real-time systems has been considered an interesting application field for automatic analysis techniques such as model-checking. Unfortunately, there is a natural trade-off between sophistication of supported features and the practicality of formal analysis.

To cope with the challenges of formal analysis real-time system designs we focus on three aspects that, we believe, are fundamental to get practical tools: model-generation, model-reduction and model-checking. Then, firstly, we extend our ideas presented in [30] and develop an automatic approach to model and verify designs of real-time systems for complex timing requirements based on scheduling theory and timed automata theory [7] (a well-known and studied formalism to model and verify timed systems). That is, to enhance practicality of formal analysis, we focus our analysis on designs adhering to Fixed-Priority scheduling. In essence, we exploit known scheduling theory to automatically derive simple and compositional formal models. To the best of our knowledge, this is the first proposal to integrate scheduling theory into the framework of automatic formal verification. To model such systems, we present I/O Timed Components, a notion and discipline to build non-blocking live timed systems. I/O Timed Components, which are build on top of Timed Automata, provide other important methodological advantages like influence detection or compositional reasoning.

Secondly, we provide a battery of automatic and rather generic abstraction techniques that, given a requirement to be analyzed, reduces the model while preserving the relevant behaviors to check it. Thus, we do not feed the verification tools with the whole model as previous formal approaches. To provide arguments about the correctness of those abstractions, we present a notion of Continuous Observational Bismulation that is weaker than strong timed bisimulation yet preserving many well-known logics for timed systems like

TCTL [3].

Finally, since we choose timed automata as formal kernel, we adapt and apply their deeply studied and developed analysis theory, as well as their practical tools. Moreover, we also describe from scratch an algorithm to model-check duration properties, a feature that is not addressed by available tools. That algorithm extends the one presented in [28].

Agradecimientos

Agradezco a las instituciones que me permitieron llevar a cabo este proyecto: el Departamento de Computación, la Facultad de Ciencias Exactas y Naturales, UBA y el proyecto FOMEC. Así mismo, extiendo mi agradecimiento a las actuales y pasadas autoridades, integrantes y ex-integrantes del Departamento de Computación quienes me apoyaron incondicionalmente. Merecen una especial mención, por su aporte y compromiso en esta tesis, mi director, el Dr. Miguel Felder, mis jurados: el Dr. Alfredo Olivero, el Dr. Mauro Pezzè y el Dr. Sergio Yovine, los miembros suplentes del jurado: el Dr. Juan Vicente Echague, y el Lic. Gabriel Baum. Tampoco puedo dejar de mencionar al coautor e inspirador del trabajo sobre propiedades de duración, el Dr. Dang Van Hung. Le doy las gracias a los miembros del Politécnico de Milan, del UNU/IIST, y del instituto VERIMAG que gentilmente me recibieron e inspiraron. Agradezco epecialmente a los "tesistas" Gabriela Finkelstein, Diego Garbervetski y Cecilia Schor por su importante contribución en la validación y corrección de parte de las ideas incluidas en esta tesis. Finalmente, por el enorme soporte emocional que me brindaron durante todos estos años, le doy las gracias a mi familia, a mi novia y a mis amigos.

Contents

List of Figures

List of Tables

Abstract

Real-time systems are found in an increasing variety of application fields. Usually, they are embedded systems controlling devices that may risk lives or damage properties: they are safety critical systems. Hard Real-Time requirements (late means wrong) make the development of such kind of systems a formidable and daunting task. The need to predict temporal behavior of critical real-time systems has encouraged the development of an useful collection of models, results and tools for analyzing schedulability of applications (e.g., [109]). However, there is no general analytical support for verifying other kind of high level timing requirements on complex software architectures. On the other hand, the verification of specifications and designs of real-time systems has been considered an interesting application field for automatic analysis techniques such as model-checking. Unfortunately, there is a natural trade-off between sophistication of supported features and the practicality of formal analysis.

To cope with the challenges of formal analysis real-time system designs we focus on three aspects that, we believe, are fundamental to get practical tools: model-generation, model-reduction and model-checking. Then, firstly, we extend our ideas presented in [30] and develop an automatic approach to model and verify designs of real-time systems for complex timing requirements based on scheduling theory and timed automata theory [7] (a well-known and studied formalism to model and verify timed systems). That is, to enhance practicality of formal analysis, we focus our analysis on designs adhering to Fixed-Priority scheduling. In essence, we exploit known scheduling theory to automatically derive simple and compositional formal models. To the best of our knowledge, this is the first proposal to integrate scheduling theory into the framework of automatic formal verification. To model such systems, we present I/O Timed Components, a notion and discipline to build non-blocking live timed systems. I/O Timed Components, which are build on top of Timed Automata, provide other important methodological advantages like influence detection or compositional reasoning.

Secondly, we provide a battery of automatic and rather generic abstraction techniques that, given a requirement to be analyzed, reduces the model while preserving the relevant behaviors to check it. Thus, we do not feed the verification tools with the whole model as previous formal approaches. To provide arguments about the correctness of those abstractions, we present a notion of Continuous Observational Bismulation that is weaker than strong timed bisimulation yet preserving many well-known logics for timed systems like

generally, involving more than one component at design level) and other safety properties "make or break" design correctness [77]. Some examples of these properties are: 1) responsiveness (bounds on response times for events); 2) timing requirements for sample and output rates; 3) freshness constraints (bounds on the age of data used in the system); 4) correlation constraints (the maximum time-skew between several inputs used to produce an output); 5) availability of data or space when asynchronous communication is performed; 6) no loss of signals; 7) guarantee on the consistency of distributed data, 8) duration properties (properties stating bounds on accumulated sojourn times in system states), etc.

The problem we address in this thesis is the verification of such kind of requirements on physical designs of RTS. In what follows, we briefly describe previous approaches and motivate our approach. The interested reader can find in Annex [33] a further detailed survey on notations and analysis tools for RTS designs.

1.1 Prior Research

The desire to predict timed behavior of RTS has encouraged the development of a useful collection of results for run time scheduling (see for example [43, 109]). Furthermore, schedulability of applications satisfying studied assumptions can be efficiently analyzed and several tools have been developed (e.g., [122, 13, 97, 93, 98], etc.). However, these results and tools are mainly aimed at verifying schedulability. There is no general support for verifying distributed architectures for high-level timing requirements whose satisfaction depends on the complex interaction among several components.

On the other hand, the verification of RTS designs has been considered an interesting application field for the research community working on automatic state-space analysis, generically called Model-Checking. Model-checking is a generic name for a set of automatic and (usually quite fast) techniques for verifying concurrent systems such as sequential circuit designs and communication protocols ([56, 95, 54],etc.). If the design contains an error, model-checking will produce a counterexample that can be used to pinpoint the source of the error. These widespread techniques (which were awarded the 1998 ACM Paris Kanellakis Award for Theory and Practice), has been used successfully in practice to verify real industrial designs, and companies are beginning to market commercial model checkers. Although there are many formal based tools to describe and analyze timed systems (see for example, [88, 33, 138]) few of them address a key issue of low level design descriptions: the fact that resources are shared by components. To address the processor-sharing phenomena (more specifically the preemption of tasks, which is commonly featured in many real-time OS platforms) some operational formal notations have been adapted and some tools have also been developed. Among others, we can mention RTSL [73], which is a process algebraic approach based on discrete time to analyze task schedulability on monoprocessor systems. The technique allows designers to define existing and new scheduling disciplines; reachability analysis can be applied to detect exceptions like missed

deadlines. We can also mention GCSR [21, 142], a graphical language based on a discrete time version of a process algebra which takes into account fixed priorities to share resources [117]. To model preemption designer should divide tasks into non-preemptable units. A further technique is VERUS [46, 47], which is based on discrete labeled edge systems to model applications running on a fixed-priority scheduled single processor. VERUS uses symbolic model-checking ([36, 156], etc.) to verify time properties and to obtain timing information.

There are other approaches like [15, 38] based on a dense-time formalisms like Constant Slope Hybrid Automata [5]. Also, with a dense-time model it is possible to build conservative abstractions of real-time software, see [58]. Unlike the so far mentioned discrete-time based approaches, time granularity becomes largely irrelevant. By using such expressive dense-time formalism, it is possible to model applications which do not necessarily satisfy the assumptions of most studied scheduling theories. The drawback is that reachability is no longer decidable (although the authors claim to obtain responses in most cases). Other authors use a decidable class of Hybrid Automata to model preemption [123] but complexity still is the big issue.

Although focused on non-preemptive scheduling, it is worth citing [70] where a general model of non-preemptive aperiodic tasks based on Timed Automata is presented. Timed Automata specifies the possible arrival times of tasks, which are described as the worst case computation time and a relative deadline. Then, schedulability analysis and other safety properties are transformed into a model-checking problem.

Finally, let us mention RTD [71] which is a family of notations based on a dense-time high-level version of Petri Nets (HLTPN [80]) inspired on POSIX standards [133]. The approach is mainly aimed at simulation, symbolic execution and bounded reachability.

In conclusion, to deal with preemption, discrete time approaches resort to the "tick" resolution modeling capabilities (see [71, 46, 21]) while dense-time formalism use "integrator" variables (see [38, 15, 71, 152]) to keep track of executed time. Unfortunately, those techniques have a negative impact on the decidability or the feasibility of the verification problems for the mentioned formal approaches (e.g., see discussions in [38, 58, 132]). Moreover, the produced models are either monolithic or they are composed by a set of terms which behaviors are heavily interdependent since they share the same processing resource. We believe that this is an unsatisfactory situation; scalability becomes a very difficult issue since the whole model of the system must be always taken into account in the verification process. Indeed, the elimination of a data independent task may dramatically change the timed behavior of the remaining tasks.

1.2 Working Examples

We focus our proposal on designs which adhere to the assumptions of Fixed-Priority scheduling theory [109, 86]. Fixed-priority scheduling has been successfully used by a

large number of applications in the RTS field [109, 12, 40, 108, 139, 66],etc. It has also inspired real-time extensions to O.S. standards like POSIX [133] and JAVA Real Time extensions [135]. In particular, we propose an automatic verification technique for concrete architectures, featuring:

- Event and time-triggered tasks distributed in a net of processing resources and scheduled by a preemptive fixed-priority policy (e.g., Rate Monotonic [109]). Data and control are communicated through shared variables, queues, circular buffers, signals, etc.
- Abstract description of tasks code (where relevant events do not necessarily occur at the end of them).
- Description of known assumptions about the environment behavior.

We aim at assessing whether complex safety properties are valid for non-trivial designs built using those features. In this thesis, we propose the basis for an efficient automatic formal approach to complement manual reasoning and simulations. In this way, more confidence would be gained on the correctness of proposed detailed designs.

In this section we present extensions of two working examples found in RTS literature. They will help us to illustrate different ideas, aspects and features of our proposal along this thesis. Also, they are useful to give an early flavor of our practical achievements. To present each example, we describe its goal, particularities, proposed design (graphically sketched with a notation based on HRT-HOOD notation [42]), the requirements, and properties that the designer would like to check. The graphical notation shows three kind of objects: cyclic, sporadic, and protected objects (the initial letters in the left corner). Cyclic objects correspond to periodic tasks (i.e., time-triggered tasks awaken periodically). Sporadic objects are tasks that respond to events arriving with a minimum interarrival time. Protected Objects are monitor-like constructs that allow up to one client object to perform an operation (the set of operations is annotated in the attached rectangle). Mutual exclusion is achieved using priority ceiling emulation scheme [109, 141], i.e., the operation code executes at a priority higher than the priorities of the client object code (see Chapter4). Dotted arrows represent signal flows while solid lines are data flow.

This thesis presents the theoretical and engineering concepts for a tool that formally checks such kind designs for their requirements.

1.2.1 The Active Structural Control System

In this section we describe the design of a well-known working example: the active structural control system (see [69, 104], etc). Active structures include an embedded system to limit structural vibration due to earthquakes or strong winds. These structures include actuators that may be expanded or contracted to counteract the external forces applied

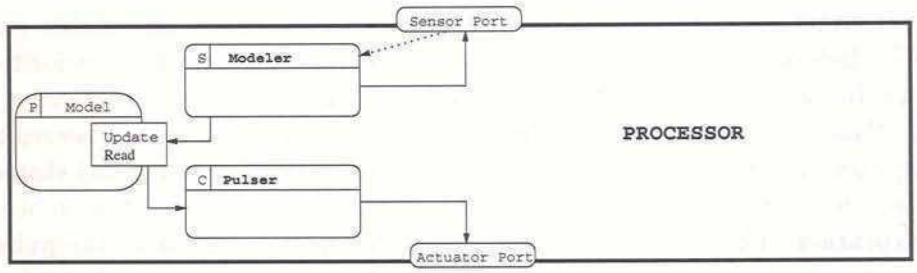


Figure 1.1: The Active Structural Control System

to the structure. A controller measures the state of the structure (e.g., accelerations and displacements) and sends commands to the actuators when readings indicate an undesirable state. To minimize vibration-induced earthquake damage, the natural frequencies of a structure should be located outside the frequency band of the seismic excitations produced by earthquakes. Then, schematically, a control system senses the seismic excitations with a high sampling rate and changes the natural frequencies of the structure by using the active members, according to the control algorithm. There are of course, timing constraints on the activity of actuators due to time bound required by the control algorithm (if not satisfied the structure may become unstable). The solution, proposed in [69] uses a pulse control algorithm. To limit vibratory displacement near resonance, those algorithms apply an opposing pulse at a higher frequency to break up the resonant forces. The mayor design variables are the time within pulse initiations Δt_p,and the pulse duration Δt. These values are determined by the natural frequencies of the system, the expected forcing functions, and the desired level of displacement control. Traditional pulse-control theory requires that the interpulse delays be of exactly the same value, namely, Δt_p. However, ensuring that these delays are invariant is almost impossible, owing to pay to "play" in factors such us the time needed for calculating the pulse magnitudes and the rump-up times for actuators. We will suppose that the structure behaves correctly if the interpulse delays are at least 32 msec * 10^{-1} but no more than 145 msec * 10^{-1} (hereafter, msec * 10^{-1} are our time units, t.u.). The following table summarizes the timing information for different system activities:

Activity	Duration (in t.u.)
Sampling	[50,55]
Pulse duration	[25,30]

Table 1.1: Computational Requirements for the Active Structural System

A socket-based communication is used. To establish a connection 10 t.u is required, when both pairs are ready to communicate it takes 5 t.u. to exchange message and acknowledgment.

Unlike the model presented in [69], we deal with a low level description of the system

where several tasks interact. That is, our solution is not a monolithic state machine like [69]. Indeed, several tasks share a single processor under fixed priority policy (see Fig. 1.1). In particular, the pulse-control algorithm is mapped into two tasks: the *Modeler* and the *Pulser*. The Modeler is a sporadic task which, once initialized, serves the messages arriving from the sensor. Then, it updates in a predictive fashion a model stored in a shared protected object *Model* and starts a new communication with the sensor to be served in the next invocation. The Pulser periodically reads the model, calculates the pulse magnitude and starts a communication with the actuator; the acknowledgement is not waited for the next iteration since the designer believes it always arrives before the next period starts (110 t.u.).

There are two main timing requirements for this system:

Regularity: As mentioned earlier, the interpulse delays should be at least 32 t.u. but no more than 145 t.u.

Freshness: The validity of the model updated by the Modeler is at most 130 t.u. That is the pulser should read a model not older than 130 t.u. (a model is "born" when the modeler updates the model object).

1.2.2 The Mine Drainage Controller System

The design of a mine drainage controller system is another example that commonly appears in the literature (e.g., [41, 102], etc.) and it possesses many of the characteristics which typify embedded real-time systems. The original presentation assumes that the system will be implemented on a single processor; however we enriched the example with a remote processor running an application that allows an operator to monitor and command the system.

We use Fig. 1.2 to explain the functionalities, design and basic timing requirements. The system is used to pump mine water, which collects in a sump at the bottom of the shaft, to the surface. The pump should not be operated when the level of methane gas (CH4) in the mine reaches a high value, due to the risk of explosion. Most environmental values - airflow, carbon monoxide (CO), and water flow - are polled periodically. High and low water sensor communicates via interrupts. The protected object *Motor* provides the services to operate the pump and reflect the motor status. The protected object *CH4 Status* keeps the level of the last methane reading. Whenever there is a risky situation (gas levels or air flow become critical, water flow readings are not consistent with motor status, etc.) an alarm is informed to a protected object, *Console*, to be eventually signaled to the remote monitor system where the operator resides. Operations and readings are added to an object *Log*. There is a sporadic task, *Command*, to serve the operator requests arriving from the remote processor: inspect motor status, set pump on or off. CO and CH4 Sensors use the technique of period displacement [41] to perform the readings. That is, they request a reading which should be available in the next period (if this were not the case, a "faulty

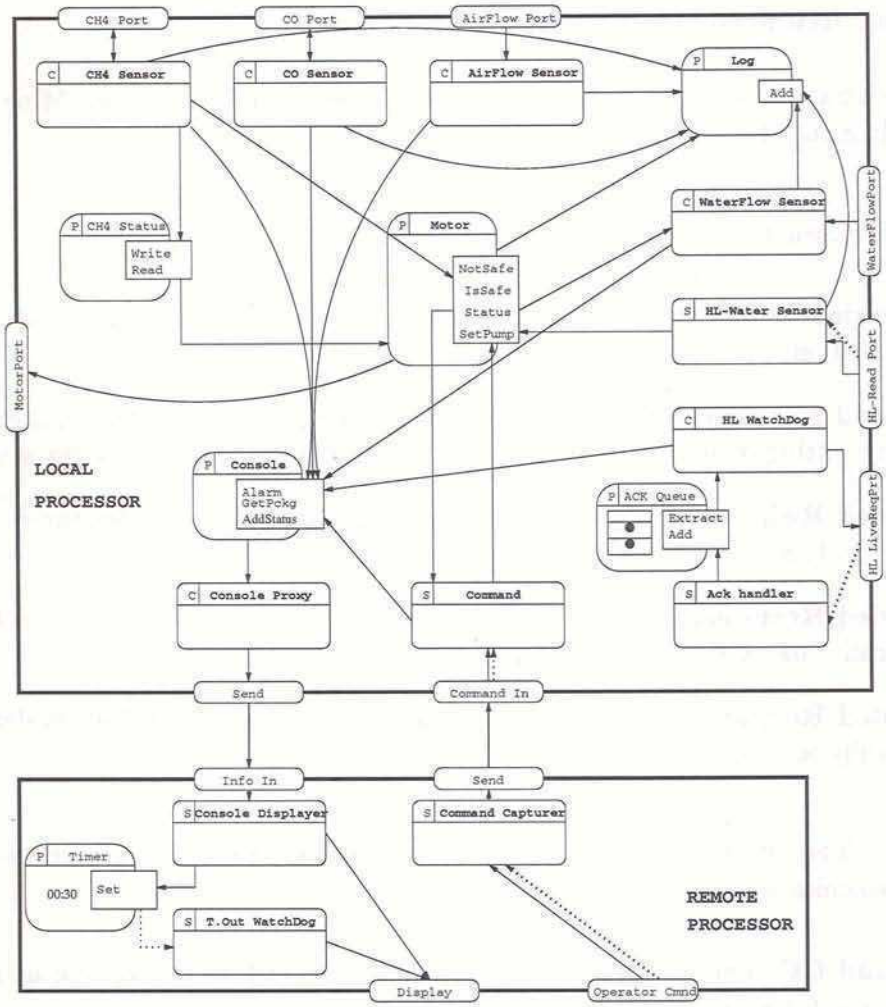


Figure 1.2: Mine Drainage Design

device" alarm is generated). We added another fault detection mechanism to the original example: a watchdog task periodically checks the availability of the high-low water sensor device. Firstly, it sends a request, and secondly, extracts the acknowledges which were received and queued in a three-slot queue by a sporadic task (the Ack Handler) in previous cycles. Finally, if the queue is empty the watchdog signals this as a faulty condition.

On the second processor, there is a sporadic task to serve the *Console Proxy* signals. The Console Proxy - which runs on the main processor - always sends a signal to indicate availability (usually, with the last values accumulated in the Console object). The *T.Out Watchdog* is awaken by a time out when no signal arrives from the proxy within 1 sec. There is an *Command Capturer* awaken by the interrupt generated from Operator Commands. This task communicates the command to its partner, *Command*, running on the local processor.

Timing Requirements and Facts

We summarize the main requirements, properties, and facts on the Mine Pump system according to [41].

Requirements:

Separation: Two consecutive readings of Water-Flow sensor shall be within 960 ms and 1,040 ms apart.

Bounded Response 1: The system must respond to high and low water level changes by setting or by clearing the motor within 200 ms (if level of CH4 is Ok).

Bounded Response 2: The deadline from CH4 going high to the pump being disabled is 200 ms.

Bounded Response 3: The operator shall be informed within 1 sec of detection of critically high CH4 or CO readings.

Bounded Response 4: The operator shall be informed within 2 sec of detection of critically low airflow readings.

Facts: The following facts are derived from the physics of the application, the sensors and transmission delays:

CH4 and CO facts: CH4 and CO sensors require at least 35 ms and at most 40 ms in order for a reading to become available.

HLW-Level Fact: There are at least 6 sec. between High and Low water level changes. The transmission of those signals takes within 10 ms and 15 ms.

We also add some new requirements and facts to illustrate some capabilities of the approach.

Requirements:

Freshness 1: The ages of CO readings and air-flow readings shall be at most 100 ms when decisions are taken in the respective sensor tasks.

Correlation: Correlation between CH4 and CO readings when written into the log shall be at most 100 ms. That is, the ages of CH4 and CO status paired into the log differ in at most 100 ms.

False Alarm: A "faulty-device" alarm shall not be informed when the devices are working properly.

Fault Detection HLW: If the high-low Water level device fails, it shall be informed to the remote operator within 2 sec.

Fault Detection NET: If a fault on the main processor or the net occurs, it shall be informed to the remote operator 2 sec.

Freshness 2: The data at CH4 status shall not be older than 100 ms.

Compliance with Object Interface Assumptions: The CH4 Sensor shall not inform more than four alarms into the time intervals between two consecutive "Get Package" Operations. Get Package is an operation performed by the console proxy to obtain the information accumulated in the console object during the last period.

Facts:

HLW-ACK Fact: When the High Low water level sensor receives the ACK request from a watchdog, the signal arrives within 15 ms and 25 ms.

Operator Fact: Operator Commands are separated at least by 1 sec.

1.3 Our Approach

To cope with the problem of checking requirements for RTS-Designs we believe it is necessary to deal with at least three kind of topics:

- Modeling Techniques.
- Model-Reduction Techniques.
- Model-Checking Techniques.

We decided to present our work guided by this threefold approach. Roughly speaking, at the modeling level, we exploit the fact that we focus on a particular (yet useful) application model. This allows us to build rather simple and verifiable models. The model-reduction techniques we present are based on a notion of influence among automata. Influence is the base for our algorithms that get rid of automata when they have no future relevance in the property satisfaction. Finally, we point out tools and algorithms for checking the real-time requirements. Moreover, we propose a novel algorithm in the case of Duration Properties ([106])(properties expressed in terms of accumulated sojourn times on particular system states).

Modeling

Motivated by the desire to improve practical feasibility of formal analysis, we decided to sacrifice the generality by fixing the architectural style of the applications to be designed. We believe that there is a natural trade-off between sophistication of supported features and the ability to efficiently verify obtained designs [30]. We chose to adhere to preemptive fixed-priority theory assumptions [109, 86]. Such assumptions suit a large number of applications in the real-time field ([109, 40, 139, 108], etc.). These assumptions allow us to exploit known scheduling theory and automatically derive simple formal models from design notations. Some scheduling analysis techniques provide analytical tools for calculating worst-case completion time of certain code areas (WCCT). Bounds for the best-case completion times (BCCT) can be derived from time estimates provided by designers. In few words, our proposal uses those calculated WCCT and BCCT to build an abstract model of the system based on Timed Automata (TA) [7] as kernel formalism. TA is a well-studied dense-time formalism simpler and more tractable than Hybrid Automata and HLTPN. TA are supported by some well established model-checking tools (e.g., [61, 23, 91, 146], etc.) which were successfully applied in some application fields such as protocol and circuit verification (e.g., [54, 59, 62, 148, 147],etc.).

In the resulting models, neither the scheduler nor the priorities of tasks are explicitly represented. However, their influence on response-times of tasks is taken into account as timing information for the TA. Indeed, we regard the scheduler and the priority assignment mainly as a way to achieve fair computation and predictive response times. Therefore, we are able to build abstractions to automatically reason on combinations of response times. We also believe that, in general, this is the kind of reasoning that a designer would rely on to gain confidence on its design. Note also that schedulability analysis is not our concern: it can be solved using scheduling theory.

In our models, a timed automaton represents each task. The edges of that timed automaton stand for the end of the task relevant actions. By using the expressive power of TA, we describe that location changes occur within the best and worst-case response times of the corresponding event (i.e., BCCT and WCCT of associated actions). The model so produced is conservative [2] as it might produce more behaviors than the actual system. Therefore, it is safe to verify safety properties (i.e., "nothing-bad-happens" properties). In real-time systems usually most interesting properties are safety properties [90]. We believe that abstracting away from scheduling details using BCCT and WCCT is faithful for most practical purposes. Indeed, in general, designers reason on worst cases to informally gain confidence on design correctness. Our approach automates that kind of reasoning. Only properties that hold on the implementation due to subtle scheduling side effects may be reported as false by our approach. In the future work section, we show how extend this approach to be more faithful adding extra information if required to check a property (e.g., harmonic periods and priorities [77] may be used to ensure tasks precedence).

[2] Conservative modeling or analysis of systems is a well-known and useful technique to reduce complexity of verification process while preserving its correctness (e.g., [52, 68, 15, 28], etc.)

Usually, designers want to know whether (and how) the system may violate the analyzed property. In our proposal, designers describe an "observer" timed-automaton, which synchronizes with those events that are relevant for checking whether the requirement is met or not (which are mainly the end of communication actions). The main kind of requirement we deal with is what we call "safety observers". A safety observer reaches a location standing for an error if and only if the system may violate the requirement.[3] TA allows designers to express rather complex timing requirements involving more than one component.

Reduction

Along with the modeling techniques we develop and prove a set of automatic reduction techniques to reduce the verification effort. For example, we present a technique to automatically derive, given a set of observable events, a subset of components that is sufficient to perform the verification process. That is, one attractive aspect of this approach is that in general, only a small subset of components is needed to perform the verification step. Hence, the complexity of our approach mainly depends on the number of components involved in the requirement instead of the size of the complete model like the previous work.

To further reduce the size of the models, we also provide a set of conservative abstraction rules. Besides, we develop a technique to reduce the size of the parallel composition of TA that uses a notion of influence to get rid of components that do not influence the future behavior of the system under analysis. Correctness of these techniques is based on extensions of bisimulation and simulation theory for timed systems we develop in this thesis.

Checking

We provide the basis, and in some cases, new algorithms to check requirements that can expressed using Safety Observers, Büchi Automata [149], Timed Computational Tree Logic [3], and Linear Duration Invariants [106]. It is worth mentioning that we resort to known tools and results to perform the analysis whenever that is possible. That is, we do not focus on the development of model-checking techniques but in the model-building and model-reduction issues. However, we present an algorithm to check Linear Duration properties and we describe as future work some improvements on the existing checking tools.

[3] Similar techniques have also been used with untimed systems like in [52, 84, 1], etc)

1.3.1 Summarizing the Achievements

One of the major contributions of this work is the integration of two research lines in real-time system verification: scheduling theory and automatic formal verification approaches.[4] Our proposal enlarges the applicability of scheduling theory to deal with requirements that involve interaction among components. We also show how known scheduling results could be used to enhance the practicality of formal analysis. The compositionality of our models implies that, generally, our analysis depends just on the components involved in the requirement; it does not require the presence of all components like previous approaches. To the best of our knowledge, it is the first proposal to model preemptive systems using a dense-time formalism which does not support time accumulation constraints (a feature supported by Hybrid Automata or HLTPN). We also provide the abstraction theory that supports the reduction methods (Chapter 3). That theory is based on notions of (somehow weak) simulation and bismulation which preserve linear and branching time structure of the original system respectively up to a set of state observers and events (and therefore, preserves the satisfaction problem for well-known logics). We introduce the notion of Timed I/O Components (Sect. 3.3) as TA where also an I/O interface is informed. That notion has nice modeling properties like non-zeno preservation, allows a rather intelligent parallel composition (Chapter 7), compositional reasoning [76, 111] and, we believe, can be applied to any other timed or untimed formalism.

Finally, we present an algorithm to model-check Duration Properties on Timed Automata. It is worth mentioning that verification of duration properties is not featured by any available tools for the analysis of real-time designs.

1.4 Structure of this Thesis

This thesis is structured as follows. Firstly, in Chapter 2,we outline the basic notions of our kernel formalism, Timed Automata, and the logics to reason on the underlying timed transition system. In Chapter 3 we show all the theoretical concepts and results that we develop to support our techniques (weak bisimulations preserving the timed branching structure, the notions of I/O Timed Components, etc.).

In our proposal, the designer describes the design and generic scenarios encompassing bad behaviors he/she wants to check are absent in the design. Chapter 4 presents notations to describe the design to be analyzed and to describe the undesired behaviors (observers). In Chapter 5 we show how the I/O Timed Components to model tasks are built. In Chapter 6 we present a set of ad-hoc procedures to faithfully reduce the size of the model. Those

[4] We should mention that [72] presents an integration of scheduling theory and program refinement. Unlike TA, timed program refinement calculus is a deductive formalism: designs are derived from the specification through sound rules. On the other hand, we automatically check designs for requirements. In their approach, schedulability tests become available as feasibility checks during system refinement while we use WCCT to build the model to be checked.

Chapter 2

Background on Timed Systems Theory

As it is shown later, we use Timed Automata (TA) as the kernel formalism to model and analyze timed behaviors. Timed automata has become one of the most widely used formalism to model and analyze timed systems. This formalism is supported by several tools (e.g., [61, 23, 146, 91], etc.) which were successfully applied to automatically check communication protocols and circuits (see [54] for some links) and are used by several research groups both in the academy and the industry.

What follows is a brief outline of the basic notion of TA (for more thorough presentations, see for example [7, 128, 61, 158, 92], etc.). We also present classical equivalence relation between timed models and two well known logics to reason on timed systems : TCTL [3] and LDI [106, 161].

2.1 Preliminar Definitions

2.1.1 Sequences

First, we introduce some basic notations for sequences that will be used in the sequel. Let s be a sequence. Then $|s|$ will denote the length of the sequence (i.e. number of elements) and s_i will denote the ith element of the sequence s, for $0 \leq i < |s|$. For $0 \leq i \leq j < |s|$, let $s_{\lfloor i}$ denote the prefix of s that ends with the ith element, $s_{[i}$ the suffix of s that starts from the ith element and $s_{[i,j]}$ the subsequence that starts from the ith element and ends with the jth element inclusively. If the sequence s is not empty, its first element (i.e., s_0) will be denoted by $first(s)$ and its last element (i.e. $s_{|s|-1}$) will be denoted by $last(s)$. The concatenation of two sequences s and s' will be denoted by ss', and a sequence with a single element will be identified with its element.

Given a set E, a subset X of E and a sequence s over E, $X \cap s$ will denote the intersection between X and the underlying set of s. $last\_X\_in\_s$ will denote the greatest natural number k such that $s_k \in X$ and $\forall l, k < l < |s|$ it holds that $s_l \notin X$ if it exists, or 0 if not.

2.1.2 Timed Words

Now we define timed languages to describe the behavior of timed automata as required in some chapters of this thesis. By *time sequence* we mean a non-decreasing sequence of non-negative real numbers.

For our convenience, we define the operation $\lhd$ between time sequences as follows. For time sequences τ and τ', $\tau \lhd \tau' \stackrel{\text{def}}{=} \tau\tau$", where τ" $\stackrel{\text{def}}{=} (\tau'_0 + last(\tau))(\tau'_1 + last(\tau)) \ldots (\tau'_{|\tau'|-1} + last(\tau))$.

Intuitively, the sequence τ' is translated by the last element of τ and is then concatenated to the sequence τ to give the result of this operation. For example, $(0\ 2\ 3\ 5.5) \lhd (1\ 5.3) = (0\ 2\ 3\ 5.5\ 6.5\ 10.8)$.

Definition 1 (Timed Word) *A (ω-)timed word over T is a pair (σ, τ), where σ is a (infinite) finite sequence of elements of T and τ is a (infinite) finite time sequence, both having the same length.*

A set of timed words over T is called a *timed language* over T.

2.1.3 Clocks, Constraints and Valuations

This presentation follows [158]. Given a finite set of variables $X = \{x_1, x_2, \ldots, x_n\}$ a *valuation* is a total function $v : X \stackrel{tot}{\rightarrow} \Re_{\geq 0}$ where $v(x_i)$ is the value associated with clock x_i.

We define V_X as the set $[X \stackrel{tot}{\rightarrow} \Re_{\geq 0}]$ of total functions mapping X to $\Re_{\geq 0}$. $\mathbf{0} \in V_X$ denotes the function that valuates to 0 all clocks. Given $v \in V_X$ and $\delta \in \Re_{\geq 0}$, $v + \delta$ denotes the valuation that assigns each clock $x \in X$ tha value $v(x) + \delta$.

Given X a set of clocks, $\rho \subseteq X$ and a valuation v we define $Reset_\rho(v)$ as:

$$Reset_\rho(v)(x) = \begin{cases} 0 & \text{si } x \in \rho, \\ v(x) & \text{otherwise.} \end{cases}$$

Given X a set of clocks we define the sets of clock constraints Ψ_X and Φ_X, according to the following grammar:

$$\Psi_X ::= x \prec c | x - x' \prec c | \psi \wedge \psi | \neg\psi$$

$$\Phi_X ::= x \leq c | \phi \wedge \phi$$

where $x, x' \in X, \prec \in <, \leq$ y $c \in I\!N$

A valuation $v \in V_X$ satisfies $\psi \in \Psi_X$ ($v \models \psi$) iff

$$\begin{array}{lll} v \models x \prec c & \iff & v(x) \prec c \\ v \models x - x' \leq c & \iff & v(x) - v(x') \leq c \\ v \models \psi \wedge \psi' & \iff & v \models \psi \wedge v \models \psi' \\ v \models \neg\psi & \iff & v \not\models \psi \end{array}$$

2.2 Timed Automata

TA are finite automata where time is incorporated by means of clocks. As finite automata, TA are composed of a finite set of nodes (called locations in TA literature) and a set of labeled edges. There is no notion of final locations since executions are infinite. Edges model event occurrences while clocks serve to measure time elapsed since these occurrences. A set of clocks (noted between braces {}) is associated with each edge to indicate which ones are reset when the edge is traversed. A timing condition - a guard- is associated with an edge. A guard is constraint on clock values. An edge may be traversed only when its guard is true, if so it is executed instantaneously and associated clocks are reset. Time elapses at locations, and edge traversal is instantaneous. Also, timing conditions are associated with each location -called an invariant- and determine the valid clock values for locations. Hence, it is possible to use an invariant to express that the control can not remain in the location more than a certain amount of time (a deadline).

Definition 2 (Timed Automata) *A timed automata is a tuple* $A =< S, X, \Sigma, E, I, s_0 >$ *where*

1. S *is a finite set of locations.*
2. X *is a finite set of clocks.*
3. Σ *is a set of labels.*
4. E *is a finite set of edges. Each edge* $e \in E$ *is a tuple* $< s, a, \psi, \alpha, s' >$ *where:*
   (a) $s \in S$ *is the source location*
   (b) $s' \in S$ *is the target location*
   (c) $a \in \Sigma$ *is a label*
   (d) $\psi \in \Psi_X$ *is the guard*
   (e) $\alpha \subseteq X$ *are the subset of clocks reset at the edge.*

5. $I : S \stackrel{tot}{\rightarrow} \Phi_X$ *is a total function associated with each location an Invariant.*

6. $s_0 \in S$ *is the initial location.*

Notation 1 *Given* $A =< S, X, \Sigma, E, I, s_0 >$ *we define:*

$Locs(A) = S$

$Clocks(A) = X$

$Edges(A) = E$

$Labels(A) = \Sigma$

$Inv(A) = I$

$Init(A) = s_0$

Given $e =< s, a, \psi, \alpha, s' > \in E$ *we define :*

$src(e) = s$

$Label(e) = a$

$Guard(e) = \psi$

$Reset(e) = \alpha$

$tgt(e) = s'$

2.3 Semantics

A *state* of the timed automaton A is a pair $(s, v) \in S \times V_X$ for which $v \models I(s)$. The set of states constitutes the *State Space* of the underlying labeled transition system of the timed automaton.

Definition 3 (Discrete Transitions) *Let* $e \in E$ *an edge. The state* $(src(e), v)$ *has a discrete transition to the state* $(tgt(e), v')$ *denoted* $(src(e), v) \mapsto_0^{Label(e)} (tgt(e), v')$ *if* $v \models Guard(e)$ *and* $v' = Reset_{Reset(e)}(v)$.

Definition 4 (Time Transitions) *Let* $t \in \Re_{\geq 0}$. *The state* (s, v) *has a time transition to* $(s, v + t)$ *denoted* $(s, v) \mapsto_t^{\lambda} (s, v + t)$ *if for all* $t' \leq t$ $v + t' \models I(s)$.

A *Labeled Transition System* (LTS) for timed systems or *State Space Graph*, $G = (Q, \mapsto_0, \mapsto^\lambda, \Sigma)$ is any graph with two types of labeled edges, discrete and time edges. Discrete edges are labeled with labels of the alphabet Σ (i.e., $\mapsto_0 \subseteq Q \times \Sigma \times Q$). Time edges are labeled in $\Re_{\geq 0}$ and satisfy the time continuity property [155] (i.e., for all $c, d \in \Re_{\geq 0}$, $p \mapsto^\lambda_{c+d} p''$ iff $p \mapsto^\lambda_c p' \mapsto^\lambda_d p''$). The elements of Q have a discrete part projected by means @, which does not change in time transitions. The semantics of TA A can be given in terms of the *LTS* of A, denoted G_A, which is a graph which has as nodes the states of A and the two types of edges correspond to the discrete and time transitions of A, resp. Notice that G_A has generally an uncountable set of nodes and uncountable branching. Usually, we need to identify an initial state of a LTS. In the case of G_A that initial state is $q_0 = (Init(A), \mathbf{0})$

Notation 2 *Given $q = (s, v)$ we denote:*

- $q + t = (s, v + t)$.
- $q^@ = s$
- $q(x_i) = v(x_i)$

2.3.1 Runs and Non Zenoness

A *run* r of A starting at q_0 is an infinite sequence $q_0 \mapsto^{a_0}_{t_0} q_1 \mapsto^{a_1}_{t_1} \ldots$ of states and transitions in G_A.

The *time of occurrence* of the n^{th} transition is equal to $\sum_{i=o}^{n-1} t_i$ and is denoted as $\tau_r(n)$. A *divergent run* is a run such that $\sum_{i=o}^{\infty} t_i = \infty$. The set of divergent runs of a TA A starting at state q is denoted $R^\infty_A(q)$.

A *timed label-sequence* accepted by a run is the timed word over $Labels(A) \cup \{\lambda\}$ obtained by projecting the labels of the transitions with its time of occurrence (i.e., given r, (σ, τ) is such that $\sigma_i = a_i$ and $\tau_i = \tau_r(i)$. The *language of labels* of the automaton A denoted as $L^\infty(A)$ is the set of timed label-word over $Labels(A) \cup \{\lambda\}$ such that have an accepting infinite divergent run starting at the initial state. Given $R \subseteq Labels(A)$ then $L^\infty_R(A)$ is the set of timed label -words of $L^\infty(A)$ filtered to show only labels that belong to R.

A *finite run* starting at state q is simply a finite sequence of states and transitions starting at q. A timed automaton is *Non-Zeno* when any finite run starting at the initial state can be extended to a divergent run that is, the set of finite runs is equal to the set of finite prefixes of divergent runs.

A timed automaton is *Strongly Non-Zeno* when any infinite run is divergent [149].

Positions

Given $r \in R_A(q)$ a position is a par $(i,t) \in I\!N \times \Re_{\geq 0}$ such that $t \leq t_i$.

We call $\Pi(r)$ the set of all positions of run r. We define a total order $<<$ on $\Pi(r)$ as follows:

$$(i,t) << (j,t') \iff i < j \vee (i = j \wedge t \leq t')$$

Given $(i,t) \in \Pi(r)$ its state is:

$$r(i,t) = q_i + t$$

The time of position $(i,t) \in \Pi(r)$, denoted $\tau_r((i,t))$ is defined as $\tau_r(i) + t$.

We say that a state w is *reachable* from state q if there exists a run $r \in R_A^\infty(q)$, and a position $(i,t) \in \Pi(r)$ such that $w = r(i,t)$.

2.3.2 From Finite Runs to Finite Transition Language

Now we define some concepts that help us in defining LDI logics (section 2.7.2). Suppose that, given a label a there is at most one edge between to locations labeled a. Then, an edge is univocally identified by its source, target locations, and its label.
The unique edge between locations s and s' labeled a will be noted as a triple $< s, a, s' >$. The first component of the triple is called the source (src), the second is the label ($label$) and the last one is the target (tgt).
Formally the set of edge identifiers E_{id} is $(Locs(A) \times Labels(A) \times Locs(A) \cup \{(\perp, init, Init(A))\}$. We will identify an edge by its source and target location avoiding its label, if there is no confusion, i.e., there is at most one edge between any pair of locations. Note that there is an element to denote the initial transition; $\perp$ is its source location while its target location is an initial location of the TA.

A timed word over E_{id} accepted by the finite run r is defined as the subsequence of discrete transitions of r and its times of occurrence where $q_n \mapsto_0^a q_{n+1}$ is converted into $< q_n^@, a, q_{n+1}^@ >$ and $< \perp, init, s_0 >$ at time 0 is the first transition.

Note that given the ith transition of a timed word (σ, τ), σ_i, we can reconstruct the value of the clocks when entering location $tgt(\sigma_i)$ after the sequence $\sigma_i]$ of transitions. The calculation is simple: the value of a clock x is $\tau_i - \tau_{last\_\delta\_in\_\sigma_i]}$, where δ is the set of transitions that reset the clock x (including σ_0). For example, let (σ, τ) be $((\perp,0)\ (0,1)\ (1,2)\ (2,3),\ 0\ 600\ 1000\ 1005)$ (a timed word accepted by finite run $(0,0) \mapsto_{600}^{\lambda} (0,600) \mapsto_0^{approach} (1,0) \mapsto_{400}^{\lambda} (1,400) \mapsto_0^{\lambda} (2,400) \mapsto_5^{\lambda} (2,405) \mapsto_0^{\lambda} (3,405)$ of the train automaton of Fig. 2.1) and let x be the clock that is reset by the transitions in $\delta = \{(\perp,0),(0,1)\}$. Then the value of the clock x when the automaton enters location

3 is $\tau_3 - \tau_{last\_\delta\_in\_\sigma_3]} = 1005 - 600 = 405$.
If a TA is no-zeno we can define the *finite transition language* of the timed automaton A simply as the set of timed words over E_{id} which have an accepting finite run. The finite transition language of A is denoted $L(A)$. The *untimed transition language* of A is a set of transition sequences σ such that there exists a time sequence τ which $(\sigma, \tau) \in L(A)$.

2.4 Parallel Composition

Given a pair of TA, A, B with disjoint set of clocks, the parallel composition $(A \parallel B)$ is built by means of the Cartesian product of their locations, the union of clocks, and the synchronization of edges by common labels. The invariant of a compound location is the conjunction of the invariants of the components. The guard of a synchronized edge is the conjunction of the local conditions, while the set of clocks to be reset is the union of the local sets. Formally:

Definition 5 (Parallel composition) *Given two TA* $A_1 =< S_1, X_1, \Sigma_1, E_1, I_1, s_{0_1} >$, *and* $A_2 =< S_2, X_2, \Sigma_2, E_2, I_2, s_{0_2} >$ *where* $X_1 \cap X_2 = \emptyset$

Let $\epsilon = \Sigma_1 \cap \Sigma_2$. *The parallel composition* $A_1 || A_2$ *is defined as:* $A =< S_1 \times S_2, X_1 \cup X_2, \Sigma_1 \cup \Sigma_2, E, I, < s_{0_1}, s_{0_2} >>$ *such that:*

- $<< s_1, s_2 >, a, \psi, \alpha, < s_1', s_2' >> \in E \iff$
  - $< s_1, a, \psi, \alpha, s_1' > \in E_1 \wedge a \notin \epsilon \wedge s_2 = s_2'$
  - $< s_2, a, \psi, \alpha, s_2' > \in E_2 \wedge a \notin \epsilon \wedge s_1 = s_1'$
  - $< s_i, a, \psi_i, \alpha_i, s_i' > \in E_i \wedge a \in \epsilon \wedge \psi = (\psi_1 \wedge \psi_2) \wedge \alpha = \alpha_1 \cup \alpha_2$
- $I(< s_1, s_2 >) = I(s_1) \wedge I(s_2)$

It is easy to see that $((s_1, s_2), v_1 v_2) \mapsto_t^l (s_1', s_2'), v_1' v_2')$ iff either $(l \in (\Sigma_1 \cap \Sigma_2) \vee t > 0)$ and $(s_i, v_i) \mapsto_t^l (s_i', v_i')$ for $(i = 1, 2)$, or $l \in \Sigma_1 - \Sigma_2$ and $(s_1, v_1) \mapsto_0^l (s_1', v_1')$ and $(s_2, v_2) \mapsto_0^\lambda (s_2', v_2') = (s_2, v_2)$, or $l \in \Sigma_2 - \Sigma_1$ and $(s_2, v_2) \mapsto_0^l (s_2', v_2')$ and $(s_1, v_1) \mapsto_0^\lambda (s_1', v_1') = (s_1, v_1)$.

It is easy to see that the $\parallel$ operator is commutative and associative. We will denote $\|_{i \in I} A_i$ the parallel composition of an indexed set of TA. If q is a state of that parallel composition $\Pi_{A_i}(q)$ will denote the local state of automaton A_i (locations and local clock values).

Let us characterize when a parallel composition of n TA can perform a discrete or time transition.

Fact 1

$$\frac{a \in \bigcup_{0 \le i \le n} \Sigma_i \wedge (\forall i) 0 \le i \le n : a \notin \Sigma_i \wedge \Pi_i(p) = \Pi_i(p') \vee \Pi_i(p) \mapsto_0^a \Pi_i(p')}{p \mapsto_0^a p'}$$

$$\frac{(\forall i) 0 \leq i \leq n : \Pi_i(p) \mapsto_t^{\lambda} \Pi_i(p')}{p \mapsto_t^{\lambda} p'}$$

That is, for a discrete transition labeled a, all TA featuring that label must have it enabled at their current local state. Then, those TA perform a a-transition while the rest remain at the same local state. For the timed transition of length t, all TA must be able to make time elapse t t.u.

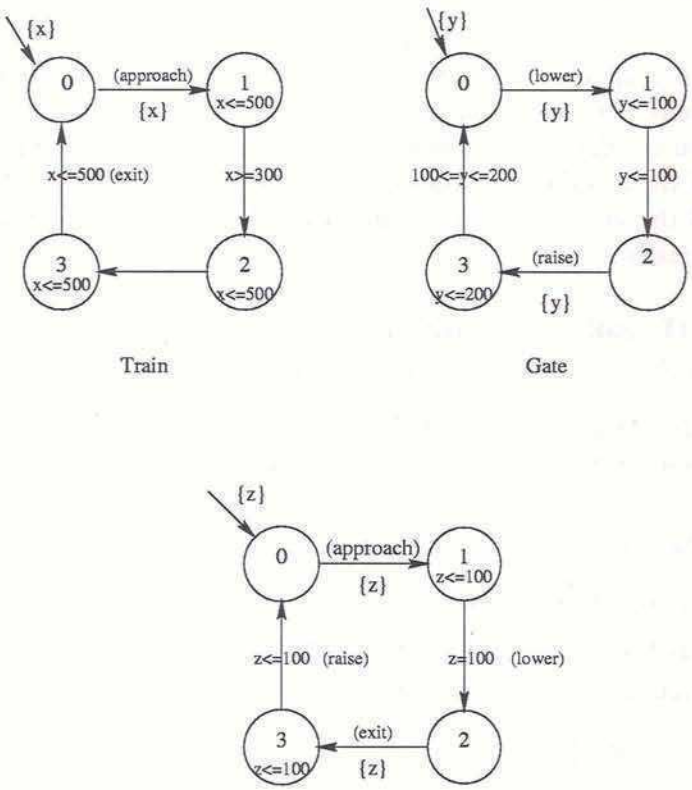


Figure 2.1: The Railroad Crossing System

Example: 1 *The figure 2.1 shows the three components of the standard Railroad Crossing System presented in [4, 104]. They are synchronized at the labels:* **approach, exit, lower, down.** *When Train approaches the crossing, it sends a signal* **approach** *to Controller and enters the crossing at least* 300 *seconds later. When Train leaves the crossing it sends a signal* **exit** *to Controller within* 500 *seconds after the signal* **approach** *has been sent. Controller sends a signal* **lower** *to the gate exactly* 100 *seconds after it has received the signal* **approach**, *and sends* **raise** *signal within* 100 *seconds after it has received the signal* **exit**. *Gate responds to the signal* **lower** *by moving down within* 100 *seconds, and responds to the signal* **raise** *by moving up between* 100 *and* 200 *seconds.*

TA can also be extended to deal with shared variables over finite domains [61, 23].

Although the problem of reachability of TA is P-SPACE hard [2], it appears to be perfectly feasibly in many interesting practical cases.[1] Several successful tools have been developed to solve reachability and other verification problems [61, 23, 146, 91], etc. In general, those tools are based on some sort of symbolic representation of the infinite state space [158].

2.5 Propositional Valuation of Locations

In order to model a real-time system, the automaton A is usually associated with a mapping $\mathcal{P} : Props \mapsto 2^{Locs(A)}$ which assigns to each location a set of propositional variables. If a location is in the image of some propositional variable then the proposition should be interpreted as *true* when the automaton stays at that location. These mappings are an abstraction mechanism to predicate on states of TA as we will show in next sections.

Definition 6 (State Valuation) *Given an LTS* $G = (Q, \mapsto_0, \mapsto^\lambda, \Sigma)$, *a valuation* $\mathcal{P} : Props \mapsto 2^{Q^@}$, *and a states* q *in* Q *then* $[q]_\mathcal{P} = \{pr \in Props / q^@ \in \mathcal{P}(pr)\}$.

Example: 2 *For the composition automaton of Fig. 2.2 (whose locations are triples indicating local locations of its three components), let* $Props = \{Up, Down, MovingUp, MovingDown, Dangerous, Trap\}$ *and* $\mathcal{P}$ *be*

$$\begin{array}{lcl} \mathcal{P}(Up) & = & \{s/\Pi_2(s) = 0\} \\ \mathcal{P}(MovingDown) & = & \{s/\Pi_2(s) = 1\} \\ \mathcal{P}(Down) & = & \{s/\Pi_2(s) = 2\} \\ \mathcal{P}(MovingUp) & = & \{s/\Pi_2(s) = 3\} \\ \mathcal{P}(Dangerous) & = & \{s/\Pi_1(s) = 1 \vee \Pi_1(s) = 2\} \\ \mathcal{P}(Trap) & = & \{s/\Pi_2(s) = 5\} \end{array}$$

2.6 Bisimulations

As we will see later, there are some relations between states of a *LTS* that imply that they are equivalent in some sense. Here, we present two well-known notions of timed-bisimulation. Bisimulations for timed systems have been studied in literature often associated to extensions of a Process Calculi (see, for instance, [137, 127, 155, 162, 115], etc.).

Definition 7 (Strong Timed-Bisimulations) *Given a LTS* $G = (Q, \mapsto_0, \mapsto^\lambda, \Sigma)$ *a symmetric binary relation* B *on* Q *is a strong timed bisimulation if* $(p, q) \in B$ *implies*

[1] However, the size of the models -locations, transitions, clocks, constants- is still a problematic issue

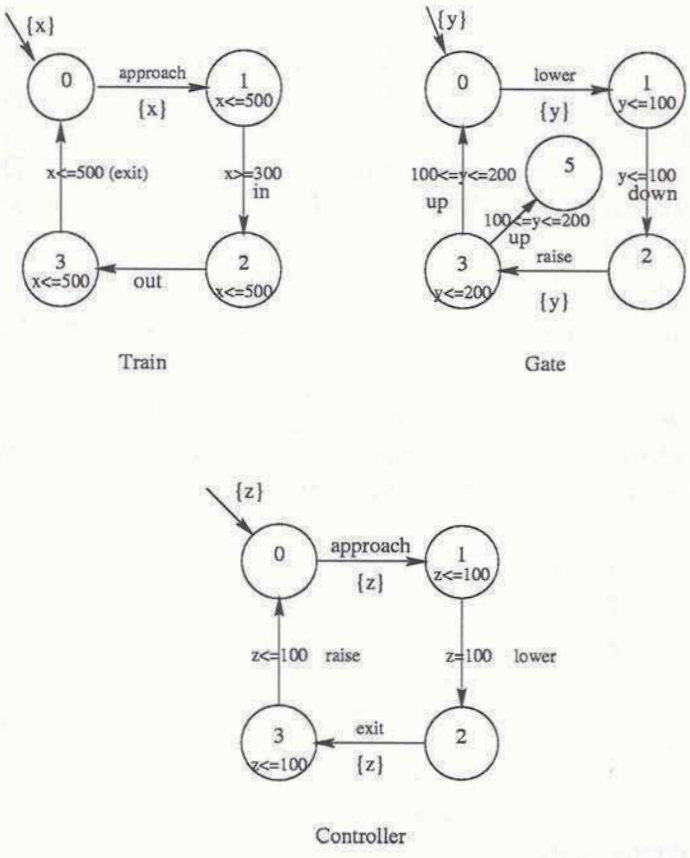


Figure 2.2: The Railroad Crossing System with a Trap Location

that for all label $a \in \Sigma \cup \{\lambda\}$ *and* $t \in \Re_{\geq 0}$ *, whenever* $p \mapsto_t^a p'$ *then, for some* q', $q \mapsto_t^a q'$ *and* $(p', q') \in B$.

As another example of relation in a LTS we have weak timed equivalence which is defined by abstracting from labels of discrete transitions [155, 162, 115].

Definition 8 *Given* $t \in \Re_{\geq 0}$, *two states* p_0, p_n *then* $p_0 \longrightarrow_t p_n$ *iff there is a finite sequence* $\sigma = p_0 \mapsto_{t_0}^{l_0} q_1 \mapsto_{t_1}^{l_1} ...p_n$ *of states and transitions such that* $\sum_{i=0}^{n-1} t_i = t$.

Definition 9 (Weak Timed Bisimulation) *Given a LTS* $G = (Q, \mapsto_0, \mapsto^\lambda, \Sigma)$ *a symmetric binary relation* B *on* Q *is a weak timed bisimulation if* $(p, q) \in B$ *implies that for all label* $a \in \Sigma \cup \{\lambda\}$ *and* $t \in \Re_{\geq 0}$ *then whenever* $p \mapsto_t^a p'$ *then, for some* q', $q \longrightarrow_t q'$, *and* $(p', q') \in B$.

2.7 Some Logics for Real-Time Models

Later in this thesis, we will deal with requirements based on two Real-Time Logics to reason on the underlying LTS: TCTL [3] and Linear Duration Invariants [106, 161]. Now these logics also illustrate the flavor of branching and linear time logics respectively.

2.7.1 Timed Computational Tree Logic

TCTL (Timed Computational Tree Logic) has been introduced in [3] as a quantitative-time extension of CTL a very well-known branching-time logic introduced by [53].

Let $\mathcal{I}$ denote a set of intervals of $\Re_{\geq 0}$ of the form $[c, c']$, $[c, c')$, $(c, c']$, (c, c'), (c, ∞) and $[c, \infty)$, where $c, c' \in I\!N$. A formula in TCTL is defined according to the following syntax:

$$\phi \ ::= \ true \ \mid \ pr \ \mid \ \neg\phi \ \mid \ \phi \vee \phi \ \mid \ \phi \exists \mathcal{U}_{\mathcal{I}} \phi \ \mid \ \phi \forall \mathcal{U}_{\mathcal{I}} \phi$$

where $pr \in Props$ is an atomic proposition and $I \in \mathcal{I}$ is an interval. Let A be a Timed Automaton. Let $\mathcal{P} : Props \mapsto 2^{Locs(A)}$ be a function associating to each atomic proposition a set of locations of A. $TCTL$ formulas are interpreted over states of G_A. Given a formula ϕ and a state q, the satisfaction relation $q \models_{\mathcal{P}} \phi$ is defined inductively on the syntax of ϕ

as follows (we ommit the subscript $\mathcal{P}$ for simplicity):

$$
\begin{array}{lll}
q \models true & & \\
q \models pr & iff & q^{@} \in \mathcal{P}(pr) \\
q \models \neg\phi & iff & not\ q \models \phi \\
q \models \phi_1 \vee \phi_2 & iff & q \models \phi_1\ or\ q \models \phi_2 \\
q \models \phi_1 \exists\mathcal{U}_{\mathcal{I}} \phi_{\in} & iff & \exists r \in R_A^{\infty}(q) and \\
 & & \exists k \in \Pi(r).\ \tau_r(k) \in I\ and\ r(k) \models \phi_2\ and \\
 & & \forall m \in \Pi(r).m << k\ then\ either\ r(m) \models \phi_1\ or\ (r(m) \models \phi_2 and\ \tau_r(m) \in I) \\
q \models \phi_1 \forall\mathcal{U}_{\mathcal{I}} \phi_{\in} & iff & \forall r \in R_A^{\infty}(q) then \\
 & & \exists k \in \Pi(r).\ \tau_r(k) \in I\ and\ r(k) \models \phi_2\ and \\
 & & \forall m \in \Pi(r).m << k\ then\ either\ r(m) \models \phi_1\ or\ (r(m) \models \phi_2 and\ \tau_r(m) \in I)
\end{array}
$$

Example: 3 *On the rail crossing system we can ask using TCTL whether whenever the train is in a dangerous state then the gate goes down within 200 time units:* $\neg Danger \vee true\forall\mathcal{U}_{[\prime,\epsilon\prime\prime]}\mathcal{D}\wr\sqsupseteq\backslash$

Some usual abbreviations are $\exists \diamond_I \phi =^{def} true\exists\mathcal{U}_{\mathcal{I}}\phi$, $\forall \diamond_I \phi =^{def} true\forall\mathcal{U}_{\mathcal{I}}\phi$, $\exists\Box_I\phi =^{def} \neg\forall \diamond_I \neg\phi$, and $\forall\Box_I\phi =^{def} \neg\exists \diamond_I \neg\phi$.

2.7.2 Linear Duration Invariants

Linear Duration Invariants (LDI in the sequel) [106, 161] is a family of real-time properties over the finite runs of timed automata. LDI formulas are Duration Calculus (DC) formulas of the form

$$\varphi \stackrel{\text{def}}{=} \sum_{b \in B} c_b \int b \sim M$$

where B is a finite set of boolean expressions over $Props$, c_b and M are reals, and $\sim \in \{\leq, <, \geq, >\}$. Given a run of the automaton A, the expression $\int b$ stands for the accumulated time in b evaluates to $true$, i.e. the time that the automaton stays in b-states. Note that accumulation is not featured neither by TA nor by TCTL.

Given a TA A, a subset of locations $F \subseteq Loc(A)$, The LDI (φ, F) is *valid* over the automaton A when $\sum_{b \in B} c_b \int b$, the duration expression, satisfies the comparison to M for those timed words accepted by finite runs of A which arrive to a location in F. (φ, F) is *satisfiable* over the automaton A when $\sum_{b \in B} c_b \int b$, the duration expression, satisfies the comparison to M over at least one timed word accepted by a finite run of A which arrives to a location in F. Clearly, an algorithm to solve validity can be used to solve satisfiability (by complementing the result of checking the negated comparison) and vice versa.

Definition 10

$$(\sigma, \tau) \models_{\mathcal{P}} \varphi \stackrel{\text{def}}{=} f^{\sigma}_{\varphi,\mathcal{P}}(\tau) \sim M$$

where

$$f^{\sigma}_{\varphi,\mathcal{P}}(\tau) \stackrel{def}{=} \sum_{1 \leq i < |\sigma|} \alpha(src(\sigma_i))(\tau_i - \tau_{i-1})$$

and $\alpha : S \rightarrow \Re$ *is the mapping assigning the contribution of each location to the linear duration expression defined as*

$$\alpha(s) \stackrel{def}{=} \sum_{\{b \in B / \bigwedge_{pr \in Props / s \in \mathcal{P}(pr)} pr \Rightarrow b\}} c_b$$

Definition 11

$$A \models_{\mathcal{P}} (\varphi, F) \stackrel{def}{=} \forall (\sigma, \tau) \in L(A) \wedge tgt(last(\sigma)) \in F \Rightarrow (\sigma, \tau) \models_{\mathcal{P}} \varphi$$

Example: 4 *Let A be the parallel composition of the rail crossing system of Fig. 2.2 and F the subset of locations of A such that validates $Trap$. Then $(2 \int (Dangerous \wedge \neg Trap) - \int ((MovingDown \vee MovingUp \vee Down) \wedge \neg Trap) \geq 0, F)$ is valid over A iff the time the crossing is disabled is not greater than two times the time the train spent in the danger zone. Note that the analyzed runs must end with the up event. The mapping α associated with the mapping $\mathcal{P}$ is*

$$\alpha((s_1, s_2, s_3)) = \begin{cases} 1 & \text{if } (s_2 \neq 5 \wedge s_2 \neq 0) \wedge (s_1 = 1 \vee s_1 = 2) \\ 2 & \text{if } s_2 = 0 \wedge (s_1 = 1 \vee s_1 = 2) \\ -1 & \text{if } (s_2 \neq 5 \wedge s_2 \neq 0) \wedge (s_1 = 0 \vee s_1 = 3) \\ 0 & \text{if } (s_2 = 0 \wedge (s_1 = 0 \vee s_1 = 3)) \vee s_2 = 5 \end{cases}$$

Let $\sigma = (\perp, \lambda, 000)\ (000, approach, 101)\ (101, lower, 112)\ (112, down, 122)$ *and* $\tau = (0\ 600\ 700\ 750)$. *Then* $f^{\sigma}_{\varphi,\mathcal{P}}(\tau) = 0 \times 600 + 2 \times 100 + 1 \times 50 = -1750$.

Since LDIs are universal properties, it is obvious that

Lemma 1 *For any pair of automata A and A' such that $L(A) \subseteq L(A')$, and $F \in Locs(A) \cap Locs(A')$ then $A' \models_{\mathcal{P}} \varphi \Rightarrow A \models_{\mathcal{P}} \varphi$.*

Proof 1 $A' \models_{\mathcal{P}} (\varphi, F) \stackrel{def}{=} \forall (\sigma, \tau) \in L(A') \wedge tgt(last(\sigma)) \in F \Rightarrow (\sigma, \tau) \models_{\mathcal{P}} \varphi$. *Since $L(A) \subseteq L(A')$ then $\forall (\sigma, \tau) \in L(A) \wedge tgt(last(\sigma)) \in F \Rightarrow (\sigma, \tau) \models_{\mathcal{P}}$ and by definition $A \models_{\mathcal{P}} (\varphi, F)$*

Chapter 3

Extensions on Timed Systems Theory

In this chapter we present the theoretical kernel of this work. Firstly, we present definitions and results which are the base of our abstraction techniques. Secondly, we present an I/O version of TA that we call I/O Timed Components which satisfy some important modeling properties.

3.1 Property Preserving Simulations

We want to define the theoretical notions that support the correctness of some of our abstraction mechanisms. In order to do that, we define when a TA can simulate another one up to a set of labels considered as relevant (or visible) and a set of propositional variables to observe states. We will see that two TA so related are indistinguishable wrt. linear time properties. These simulations are inspired on the idea of observational timed-simulation found elsewhere ([155, 111],etc.) and state-event observers [116] for discrete time, and takes into account not only visible events but propositional assignment as well.

Definition 12 *Let G an LTS of a timed system, $Q^{@}$ the discrete locations of G. Given $t \in \Re_{\geq 0}$, a set of labels R, Props a set of propositional variables, and $\mathcal{P} : Props \mapsto 2^{Q^{@}}$ a propositional assignment to locations. The state q_0 has time transition to q_n up to R and $\mathcal{P}$ denoted $q_0 \xrightarrow{R,\mathcal{P}}_t q_n$ if and only if there is a finite sequence $q_0 \mapsto^{a_0}_{t_0} q_1 \mapsto^{a_1}_{t_1} ...q_n$ of states and transitions such that $\sum_{i=0}^{n-1} t_i = t$ and for all $0 \leq i < n : a_i \notin R$ and $[q_0]_{\mathcal{P}} = [q_i]_{\mathcal{P}}$ $(1 \leq i \leq n)$ (remember $\mapsto^{a_i}_{t_i}$ could be $\mapsto^{\lambda}_{0}$ i.e., a stutter).*

Definition 13 (Simulation) *Given two TA A_1 and A_2, a set of labels R, Props a set of propositional variables, and $\mathcal{P}_i : Props \mapsto 2^{Q_{A_i}{}^{@}}$ $(i = 1, 2)$ a propositional assignment to*

locations of Q_{A_1} and Q_{A_2} resp.; a relation S_R between the state space G_{A_1} and the state space of G_{A_2} (i.e., $S_R \subseteq Q_{A_1} \times Q_{A_2}$) is a simulation w.r.t. R and $\mathcal{P}_i$ $(i = 1,2)$ iff for all pairs $(p,q) \in S_R$, then $[p]_{\mathcal{P}_1} = [q]_{\mathcal{P}_2}$ and for all $a \in Labels(A) \cup \{\lambda\}$ and $t \in \Re_{\geq 0}$ the following conditions hold:

- *whenever $p \mapsto_0^a p'$ and $a \in R$ then, for some q', q_1, q_2, $q \xrightarrow{R,\mathcal{P}_2}_0 q_1 \mapsto_0^a q_2 \xrightarrow{R,\mathcal{P}_2}_0 q'$ and $(p',q') \in S_R$, and*
- *whenever $p \mapsto_t^a p'$ and $a \notin R$ then, for some q', $q \xrightarrow{R,\mathcal{P}_2}_t q'$ and $(p',q') \in S_R$.*

A timed automaton A_2 simulates A_1 up to R and $\mathcal{P}_i$ $(i = 1,2)$ denoted $A_1 \preceq_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$ if and only if there exists a relation $S_R \subseteq Q_{A_1} \times Q_{A_2}$ a simulation w.r.t. $R, \mathcal{P}_i$ $(i = 1,2)$ such that the initial states are related by S_R. By using the identity relation, it is easy to see that $\preceq_R^{\mathcal{P}_1,\mathcal{P}_2}$ is reflexive (i.e., $A \preceq_R^{\mathcal{P}_1,\mathcal{P}_1} A$) and, by using the composition of simulations, it can be proved that it is a transitive relation (i.e. $A_1 \preceq_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$ and $A_2 \preceq_R^{\mathcal{P}_1,\mathcal{P}_3} A_3$ then $A_1 \preceq_R^{\mathcal{P}_1,\mathcal{P}_3} A_3$).

We say that two TA A_1 and A_2 are *simulation equivalent* up to $R, \mathcal{P}_i$ $(i = 1,2)$ denoted $A_1 \approx_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$ if and only if $A_1 \preceq_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$ and $A_2 \preceq_R^{\mathcal{P}_2,\mathcal{P}_1} A_1$.

Notation 3 *Whenever the propositional assignment has no importance $Props = \emptyset$ (states are unobservable) we just omit it as a superscript. The same criteria is applied when the set of labels R is the empty set.*

The following results hold:

- If $A_1 \preceq_R A_2$ then for any run of A_1 there is run of A_2 such that shows the same R -labels with the same occurrence times (ie., $L_R^\infty(A_1) \subseteq L_R^\infty(A_2)$).
- $(A \parallel C) \preceq_{Labels(A)} A$ holds.
- On the other hand, $A \preceq_{Labels(A)} (A \parallel C)$ holds if C has the following property: for all $a \in Labels(A)$ for all reachable state q, there exists a state q' such that $q \mapsto_0^a q'$ (non-blocking).

Lemma 2 (Congruence) *Given three TA A_1, A_2, and C, such that $A_1 \preceq_R A_2$ and $Labels(A_1) \cap Labels(C) = Labels(A_2) \cap Labels(C) \subseteq R$, and $\mathcal{P} : Props \mapsto 2^{Locs(C)}$ then $C \parallel A_1 \preceq_R^{\mathcal{P}_1,\mathcal{P}_2} C \parallel A_2$ where $\mathcal{P}_i (i = 1,2)$ are the natural extensions of P to the respective cartesian products.*

Proof 2 *Let S_R be the simulation between A_1 and A_2. We extend the simulation to $A_1 \parallel C$ and $A_2 \parallel C$ by also requiring that the C part of the state should be identical for related*

states (i.e., $(p,q) \in S*_R$ iff $\Pi_{A_1}(p) S_R \Pi_{A_2}(q)$ and $\Pi_C(p) = \Pi_C(q)$). This is indeed a simulation: In fact, let $(p,q) \in S*_R$, firstly, note that, due to the characterization of the parallel composition, any discrete transition in $G_{A_1 \| C}$ from state p to state p' is a discrete transition of A_1 and/or a discrete transition of C. If its just a local transition of C it follows that q can also perform the jump to a bisimilar state. If $\Pi_{A_1}(p) \mapsto_0^a \Pi_{A_1}(p')$ and $a \in R$ then for some q', q_1, q_2, $\Pi_{A_2}(q) \xrightarrow{R}_0 q_1 \mapsto_0^a q_2 \xrightarrow{R}_0 \Pi_{A_2}(q')$ and $(p',q') \in S*_R$. From the fact that non relevant labels do not synchronize and $\Pi_C(q)$ can stutter and perform the same discrete jump that $\Pi_C(p)$, we conclude that $q \xrightarrow{R}_0 s_1 \mapsto_0^a s_2 \xrightarrow{R}_0 q'$ and $(p',q') \in S*_R$.

By using similar arguments, if $p \mapsto_t^a p'$ and $a \notin R$ then it is easy to see that both projections can perform non-synchronized runs and for some q', $q \xrightarrow{R}_t q'$ and $(p',q') \in S*_R$.

It follows that

Corollary 1 *Given three TA A_i $(i = 1,2)$ and O, a labeling function $\mathcal{P} : Props \mapsto 2^{Locs(O)}$ if $A_1 \approx_{Labels(O)} A_2$ (simulation equivalent) then $A_1 \| O \approx^{\mathcal{P}_1,\mathcal{P}_2} A_2 \| O$ where $\mathcal{P}_i (i = 1,2)$ are the natural extensions of $\mathcal{P}$ to the respective cartesian products.*

Theorem 1 *Given two TA A_i $(i = 1,2)$, a function associating to each atomic proposition a set of locations of A_i $\mathcal{P}_i : Props \mapsto 2^{Locs(A_i)}$ $(i = 1,2)$. Assume that $A_1 \preceq^{\mathcal{P}_1,\mathcal{P}_2} A_2$ then for any reachable state $p \in G_{A_1}$ there exists a reachable state $q \in G_{A_2}$ such that $[p]_{\mathcal{P}_1} = [q]_{\mathcal{P}_2}$.*

Proof 3 *Naturally, simulations means that we can mimic runs at the propositional level. That is, given any finite run $r_1 = q_0 \mapsto_{t_0}^{a_0} q_1 \mapsto_{t_1}^{a_1} ...q_n$ we can find a simulation of it, namely r_2, of the same time length by applying n- times the simulation definition.*

Not only reachability is preserved through simulations; in general, logics based on linear time are preserved through simulations. Now, we will see that LDI is preserved through simulations.

Theorem 2 *Given two TA A_i $(i = 1,2)$, a function associating to each atomic proposition a set of locations of A_i $\mathcal{P}_i : Props \mapsto 2^{Locs(A_i)}$ $(i = 1,2)$. Let ϕ be a LDI-formula on Props. Assume that $A_1 \approx^{\mathcal{P}_1,\mathcal{P}_2} A_2$ then $A_1 \models_{\mathcal{P}_1} \phi \iff A_2 \models_{\mathcal{P}_2} \phi$.*

Proof 4 *Simulations means that we can mimic runs at the propositional level. That is, given any finite run r_1 we can find a simulation of it, namely r_2, of the same time length by applying repeatedly the simulation definition. In its clear that, the timed words accepted by r_1 and r_2 when projected into the transitions which really change the propositional assignment exhibit the same time sequence. Therefore the LDI has the truth value (see Sect. 2.7.2).*

From the last Theorem and Corollary 1 it follows:

Corollary 2 *Given three TA A_i $(i = 1,2)$ and O, a labeling function $\mathcal{P}$: Props $\mapsto$ $2^{Locs(O)}$, if $A_1 \approx_{Labels(O)} A_2$ (simulation equivalent) then for all ϕ LDI, $A_1 \parallel O \models_{\mathcal{P}_1} \phi$ and $A_2 \parallel O \models_{\mathcal{P}_2} \phi$ where $\mathcal{P}_i (i = 1,2)$ are the natural extensions of $\mathcal{P}$ to the respective cartesian products.*

3.2 Property Preserving Bisimulations: CO-Bisimulations

In this section, we present a notion of bisimulation which is weaker notion than the strong timed-bisimulation (see Chapter 2) and it still preserves the branching structure of a timed system. It is important to note that, traditional weak bisimulations like the one presented in Definition 9 respecting the propositional assignment (i.e., $(p,q) \in B$ then $[p]_{\mathcal{P}} = [q]_{\mathcal{P}}$) do not necessarily preserve TCTL. Actually, it does not preserve the branching structure. This fact is well-known for the untimed case where there is a concept to solve this problem called "branching bisimulation" [81, 65]. In [157] there is a proposal for timed systems based on that branching bisimulation. Here we present a dual notion taking into account events and states. He proves TCTL preservation for a subset of timed systems (systems which states do not have a timed and a discrete jump simultaneously enabled). Roughly speaking, the main result of this section states that what we define as two Continuos Observational Bisimilar timed systems satisfy the same set of TCTL formula. In the next chapters, we show how to automatically build abstractions which are CO-bisimilar systems. Thanks to the results shown in this chapter, we are able to conclude that the abstractions are exact up to TCTL satisfaction.

Definition 14 *Given G an LTS of a timed system. Given $t \in \Re_{\geq 0}$, $R \subseteq \Sigma$, a relation between states B, two states q_0 and p such that $(p, q_0) \in B$. The state q_0 has a observationally-τ transition wrt. B, p and R, of lenght t to q_n , denoted $q_0 \xrightarrow{B,p,R}_t q_n$ iff there is a finite sequence $r = q_0 \mapsto^{l_0}_{t_0} q_1 \mapsto^{l_1}_{t_1} ...q_n$ of states and transitions such that $\sum_{i=0}^{n-1} t_i = t$, for every position $k \in \Pi(r)$, such that $p \mapsto^{\lambda}_{\tau_r(k)}$ then $(p + \tau_r(k), r(k)) \in B$, and $l_i \notin R$ $(i = 1..n)$ (remember $\mapsto^{a_i}_{t_i}$ could be $\mapsto^{\lambda}_0$ i.e., a stutter).*

Definition 15 (Continuous Observational Bismulations) *Given a LTS $G = (Q, \mapsto_0, \mapsto^{\lambda}, \Sigma)$ and $R \subseteq \Sigma$, a propositional assigment $\mathcal{P}$: Props $\mapsto 2^{Q^{\mathbb{Q}}}$ then a symmetric binary relation B on Q is a continuous observational bisimulation (CO-Bisimulation) wrt. R and $\mathcal{P}$ if $(p,q) \in B$ implies that $[p]_{\mathcal{P}} = [q]_{\mathcal{P}}$ and for all $a \in \Sigma$, $t \in R_{\geq 0}$,*

whenever $p \mapsto^a_0 p'$ and $a \notin R$ then, for some $q', q'' \in Q, a' \in \Sigma \cup \{\lambda\} - R$, $q \xrightarrow{B,p,R}_0 q' \mapsto^{a'}_0 q''$, and $(p', q'') \in B$,

whenever $p \mapsto^a_0 p'$ and $a \in R$ then, for some $q', q'' \in Q$, $q \xrightarrow{B,p,R}_0 q' \mapsto^a_0 q''$, and $(p', q'') \in B$,

whenever $p \mapsto_t^{\lambda} p'$ *then, for some* $q' \in Q$, $q \overset{B,p,R}{\longrightarrow}_t q'$ *(which also means that* $(p', q') \in B$.

Two TA A_1 and A_2 are *Continuous Observational bisimilar* (CO-Bisimilar) wrt. R and $\mathcal{P}_i (i = 1, 2)$ ($A_1 \simeq_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$) iff there exists a continuous observational bismulation wrt. R and $\mathcal{P}_1 \cup \mathcal{P}_2$ in the union of their LTSs, G_{A_1} and G_{A_2}, such that their initial states are bisimilar.

Notation 4 *If the propositional assignment has no importance, (i.e.,* $Props = \emptyset$ *all states are identical) we just omit it as a superscript. The same criterion is applied when the set of labels* R *is the empty set.*

Note that since the definition of CO-bisimilarity is more restrictive than simulation equivalence (the definition requires B and B^{-1} to be a particular case of simulations, that is simulations that hold at every time), $A_1 \simeq_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$, then $A_1 \approx_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$.

As simulation equivalence, CO-bismilarity is an equivalence relation: that is, reflexive (identity relation), symmetric (the relation itself is symmetric) and transitive relation (i.e. $A_1 \simeq_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$ and $A_2 \simeq_R^{\mathcal{P}_1,\mathcal{P}_3} A_3$ then $A_1 \simeq_R^{\mathcal{P}_1,\mathcal{P}_3} A_3$, composing relations).

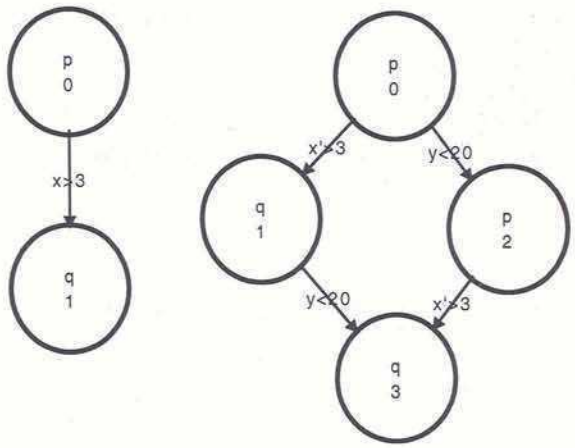


Figure 3.1: Two Continuous Observational Equivalent Automata

Example: 5 *In the example of Fig. 3.2, the CO-bisimulation is* $(s, x)B(s', x', y)$ *iff* $x = x'$ *and* $(s = 0 \wedge (S' = 0 \vee s' = 2)$ *or* $s = 1 \wedge (s' = 1 \vee s' = 3))$.

Let's prove the transitivity property:

Proof 5 *The proof is cumbersome but straightforward.*

We know that $A_1 \simeq_R^{\mathcal{P}_1,\mathcal{P}_2} A_2$ *and* $A_2 \simeq_R^{\mathcal{P}_1,\mathcal{P}_3} A_3$. *Let* B_{12} *and* B_{23} *be the bisimulation relations resp. We define* B_{13}, *or shortly* B, *as:* $(p_1,p_3) \in B \iff \exists (p_1,p_2) \in B_{12} \wedge (p_2,p_3) \in B_{23}$. B *is symmetric since* B_{12} *and* B_{23} *are so. Also* $[p_1]_{\mathcal{P}} = [p_2]_{\mathcal{P}} = [p_3]_{\mathcal{P}}$, *where* $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$.

Now, if $p_1 \mapsto_0^a p_1'$ *and* $a \notin R$, *then* $p_2 \overset{B_{12},p_1,R}{\longrightarrow}_0 p_2'' \mapsto_0^{a'} p_2'$, *with* $a' \notin R$, *and* $(p_1', p_2') \in B_{12}$. *Note that* $(p_1, p_2'') \in B_{12}$. *By a Lemma (I), to be proved afterwards,* $p_3 \overset{B,p_1,R}{\longrightarrow}_0 p_3''$ *and* $(P_2'', p_3'') \in B_{23}$, *then* $p_3'' \overset{B_{23},p_2'',R}{\longrightarrow}_0 w_3 \mapsto_0^{a''} p_3'$, *such that* $a'' \notin R$, *and* $(p_2', p_3') \in B_{23}$. *By Lemma(II)* $p_3'' \overset{B,p_1,R}{\longrightarrow}_0 w_3$. *Thus, putting al the results together:* $p_3 \overset{B,p_1,R}{\longrightarrow}_0 p_3'' \overset{B,p_1,R}{\longrightarrow}_0 w_3 \mapsto_0^{a''} p_3'$, *such that* $(p_2', p_3') \in B_{23}$, *and finally* $p_3 \overset{B,p_1,R}{\longrightarrow}_0 w_3 \mapsto_0^{a''} p_3'$, *such that* $a'' \notin R$, *and* $(p_1', p_3') \in B$.

The case $a \in R$ *is similar.*

Now, if $p_1 \mapsto_t^{\lambda} p_1'$, *then we know that* $p_2 \overset{B_{12},p_1,R}{\longrightarrow}_t p_2'$, *such that* $(p_1', p_2') \in B_{12}$ *and by Lemma (I)* $p_3 \overset{B,p_1,R}{\longrightarrow}_t p_3'$ *such that* $(p_2', p_3') \in B_{23}$ *and then* $(p_1', p_3') \in B$.

It remains to state and prove Lemma I and Lemma II:

Lemma I *Suppose that* $p_1, p_1 + t \in Q$, $(p_1, p_2) \in B_{12}$, $(p_2, p_3) \in B_{23}$. *If* $p_2 \overset{B_{12},p_1,R}{\longrightarrow}_t p_2'$, *then* $P_3 \overset{B,p_1,R}{\longrightarrow}_t p_3'$ *and* $(p_2', p_3') \in B_{23}$.

We know that $r = p_2 \mapsto_{t_0}^{l_0} q_1 \mapsto_{t_1}^{l_1} \ldots q_n = p_2'$, *such that* $\sum_{i=0}^{n-1} t_i = t$, *and for every position* $k \in \Pi(r)$, *then* $(p_1 + \tau_r(k), r(k)) \in B_{12}$, *and* $l_i \notin R$ *(for* $i = 1..n$*).*

We can prove it by induction:

- *Case* $n = 1$ *and* $t_0 = 0$: $p_2 \mapsto_0^{l_0} q_1 = p_2'$, *since* $(p_2, p_3) \in B_{23}$ *then,* $p_3 \overset{B_{23},p_2,R}{\longrightarrow}_0 w_3' \mapsto_0^{l_0'} p_3'$ *and* $(p_2', p_3') \in B_{23}$. *Since,* $(p_1, p_2') \in B_{12}$, *then* $(p_1, p_3') \in B$, *then by Lemma (II)* $p_3 \overset{B,p_1,R}{\longrightarrow}_0 w_3'$, *therefore* $p_3 \overset{B,p_1,R}{\longrightarrow}_0 p_3'$ *and* $(p_2', p_3') \in B_{23}$.

- *Case* $n = 1$ *and* $t_0 > 0$*)* $p_2 \mapsto_{t_0}^{\lambda} q_1 = p_2'$; *then we know that* $p_3 \overset{B_{23},p_2,R}{\longrightarrow}_{t_0} p_3'$ *and* $(p_2', p_3') \in B_{23}$. *That means, then by Lemma (II)* $p_3 \overset{B,p_1,R}{\longrightarrow}_{t_0} p_3'$.

- *Case* $n + 1$, $l_0 \neq \lambda$, *and* $t_0 = 0$*) Then* $r = p_2 \mapsto_0^{l_0} q_1 \overset{B_{12},p_1,R}{\longrightarrow}_t q_{n+1} = p_2'$, *and we know that* $(p_1, q_1) \in B_{12}$. *On the other hand,* $p_3 \overset{B_{23},p_2,R}{\longrightarrow}_0 q_1^{3\prime} \mapsto_0^{l_0'} q_1^3$, *such that* $(q_1, q_1^3) \in B_{23}$, *and* $l_0' \notin R$. *Then, by Lemma (II) and the fact that* $(p_1, q_1) \in B_{12}$, $p_3 \overset{B,p,R}{\longrightarrow}_0 q_1^3$. *Now, we have also that* $(p_1, q_1) \in B_{12}$ *and we have* $(q_1, q_1^3) \in B_{23}$, *then we can apply the inductive hypothesis we got* $q_1^3 \overset{B,p_1,R}{\longrightarrow}_t w$, *such that* $(q_{n+1}, w) \in B_{23}$. *Joining the results,* $p_3 \overset{B,p_1,R}{\longrightarrow}_0 q_1^3 \overset{B,p_1,R}{\longrightarrow}_t w$ *,and* $(q_{n+1}, w) \in B_{23}$ *and thus,* $p_3 \overset{B,p_1,R}{\longrightarrow}_t w$ *with* $(p_2', w) \in B_{23}$.

- *Case $n+1$, $l_0 = \lambda$, and $t_0 > 0$) Then $r = p_2 \mapsto^{\lambda}_{t_0} q_1 \overset{B_{12},(p_1+t_0),R}{\longrightarrow}_{t-t_0} q_{n+1} = p'_2$, and we know that $(p_1 + t, q_1) \in B_{12}$. On the other hand, $p_3 \overset{B_{23},p_2,R}{\longrightarrow}_{t_0} q_1^3$, such that $(p_2 + t_0, q_1^3) \in B_{23}$. Then, by Lemma (II) $p_3 \overset{B,p_1,R}{\longrightarrow}_{t_0} q_1^3$. Now, we have also that $(p_1+t, q_1 = p_2+t) \in B_{12}$ and we have $(p_2+t, q_1^3) \in B_{23}$, then $(p_1+t, q_1^3) \in B$. Applying the inductive hypothesis and we got $q_1^3 \overset{B,p_1+t_0,R}{\longrightarrow}_{t-t_0} w$, such that $(q_{n+1}, w) \in B_{23}$. Joining the results, $p_3 \overset{B,p,R}{\longrightarrow}_{t_0} q_1^3 \overset{B,p_1+t_0,R}{\longrightarrow}_{t-t_0} w$,and $(q_{n+1}, w) \in B_{23}$ and thus, $p_3 \overset{B,p,R}{\longrightarrow}_{t} w$ with $(p'_2, w) \in B_{23}$.*

Lemma II: *Suppose that $p_1, p_1 + t \in Q$, $(p_1 + t', p_2 + t') \in B_{12}$ for $t' \leq t$, and $(p_2, p_3) \in B_{23}$. If $p_3 \overset{B_{23},p_2,R}{\longrightarrow}_{t} p'_3$, then $p_3 \overset{B,p_1,R}{\longrightarrow}_{t} p'_3$.*

Let $r = p_3 \mapsto^{l_0}_{t_0} q_1 \mapsto^{l_1}_{t_1} ...q_n = p'_3$, such that $\sum_{i=0}^{n-1} t_i = t$, and for every position $k \in \Pi(r)$, then $(p_2 + \tau_r(k), r(k)) \in B_{23}$, and $l_i \notin R$ (for $i = 1..n$). Since $(p_1 + t', p_2 + t') \in B_{12}$ for $t' \leq t$ then $(p_1 + \tau_r(k), r(k)) \in B$, and therefore $p_3 \overset{B,p_1,R}{\longrightarrow}_{t} p'_3$

Now, we will prove a preservation result for TCTL following the scheme for bisimulation and CTL^* (e.g., [56]).

Definition 16 (Correspondence) *Given a LTS G and a bisimulation B. We say that a run r' of G is in correspondence with another run r of G if an only if there exists a total surjective mapping C: from positions of r' to positions of r, $C : \Pi(r') \mapsto \Pi(r)$, such that:*

- *For every $k \in \Pi(r')$ then $(r(C(k)), r'(k)) \in B$ and $\tau_{r'}(k) = \tau_r(C(k))$ (time and bisimulation preserving).*
- *$k_1 <<_{r'} k_2$, then $C(k_1) <<_r C(k_2)$ (monotone).*

Note that $C((0,0)) = (0,0)$ due to surjectivity and monotony.

Lemma 3 *If p and q are two states CO-bisimilar states then for every run r starting from p there is a run r' starting from q which is in correspondence with r.*

Proof 6 *Let $r = p \mapsto^{l_0}_{t_0} p_1 \mapsto^{l_1}_{t_1} ....$ Due to the definition of CO-bismulation if $t_0 = 0$ then $q \overset{B,p,R}{\longrightarrow}_0 w \mapsto^{a'}_0 q'_1$, that is $r'_1 = q_0 \mapsto^{l_0}_0 q_1 \mapsto^{l_1}_0 ...q_n = w$ and for every position $k \in \Pi(r'_1)$ then $(p, r'_1(k)) \in B$, and $(p_1, q'_1) \in B$. Then we can define $C((i,0)) = (0,0)$ for $0 \leq i \leq n$ and $C((n+1,0)) = (1,0)$ (that is, we map all the positions of this sequence but the last one to the first position of r and the last position to the second position of r).*

If $t_0 > 0$ a similar procedure may be applied. $q \overset{B,p,R}{\longrightarrow}_t q'_1$, that is $r'_1 = q_0 \mapsto^{l_0}_{t'_0} q_1 \mapsto^{l_1}_{t'_1} ...q_n$ where $\sum_{i=0}^{n-1} t'_i = t_0$, and for every position $k \in \Pi(r'_1)$ then $(p + \tau_{r'_1}(k), r'_1(k)) \in B$. In

this case let $C(k)$ be $(0, \tau_{r'_1}(k))$. Therefore, we started to build r' and the correspondence with the first transition of r. The least fixed point of this procedure is a sequence r' in correspondence with r.

Lemma 4 *Given the LTS of a TA A G_A, and a propositional valuation $\mathcal{P}$: Props $\mapsto$ $2^{Locs(A)}$, and a TCTL-formula on Props ϕ. Let B a continuos observational bisimulation wrt. $\emptyset, \mathcal{P}$. Assume that q, q' are bisimilar states. Then, $q \models_{\mathcal{P}} \phi \iff q' \models_{\mathcal{P}} \phi$.*

Proof 7 *We prove the lemma by induction on the structure of ϕ. Base:*

$q \models_{\mathcal{P}} pr$ iff $q^{@} \in \mathcal{P}(pr)$ iff $q'^{@} \in \mathcal{P}(pr)$ (due to bisimulation respects prop. assignment) iff $q' \models_{\mathcal{P}} pr$.

Induction: There are several cases. All propositional cases are straightforward.

$q \models \exists \phi_1 \mathcal{U}_{\mathcal{I}} \phi_{\in}$ iff $\exists r = q \mapsto_t^l ... \in R^{\infty}(q)$. and $\exists k \in \Pi(r)$: such that $\tau_r(k) \in I$ and $r(k) \models \phi_2$ $\forall m \in \Pi(r).m << k$ either $r(m) \models \phi_1$ or $(r(m) \models \phi_2$ and $\tau_r(m) \in I)$. We know that there exists a $r' = q' \mapsto_{t'}^{l'} ... \in R^{\infty}(q')$ in correspondence with r (i.e. there is an order preserving and surjective function C from the positions of r' to the position of r such that it relates bisimilar states). In particular, there exists a position k' of r' such that $C(k') = k$. Then, $(r'(k'), r(k)) \in B$ and $\tau_{r'}(k') = \tau_r(k) \in I$ and also, by inductive hypothesis, $r(k') \models \phi_2$. Moreover, all $m << k'$ $(r'(m), r'(C(m))) \in B$ and since $C(m) << C(k') = k$ then, by inductive hypothesis, $r'(m) \models \phi_1$ (in case that $r(C(m)) \models \phi_1$) or $(r'(m) \models \phi_2$ and $\tau_{r'}(m) \in I)$ otherwise, since $\tau'_r(m) = \tau_r(C(m)) \in I$. Thus, $q' \models \exists \phi_1 \mathcal{U}_{\mathcal{I}} \phi_{\in}$. Note that, this argument is symmetric.

$q \models \forall \phi_1 \mathcal{U}_{\mathcal{I}} \phi_{\in}$ iff $\forall r = q \mapsto_t^l ...s.t. \in R^{\infty}(q)$ and $\exists k \in \Pi(r)$: such that $\tau_r(k) \in I$ and $r(k) \models$ ϕ_2 $\forall m \in \Pi(r).m << k$ either $r(m) \models \phi_1$ or $(r(m) \models \phi_2$ and $\tau_r(m) \in I)$. Similarly to the previous case, we can see that every run leaving q' has a corresponding run leaving q which satisfies the formula. This argument is symmetric.

Then it is easy to conclude that two bisimilar states satisfy the same set of TCTL formula.

Theorem 3 *Given two TA A_i (i=1,2), a function associating to each atomic proposition a set of locations of A_i (i.e. $\mathcal{P}_i$: Props $\mapsto$ $2^{Locs(A_i)}$) and a TCTL-formula on Props ϕ. Assume that $A_1 \simeq^{\mathcal{P}_1, \mathcal{P}_2} A_2$ then $A_1 \models_{\mathcal{P}_1} \phi \iff A_2 \models_{\mathcal{P}_2} \phi$.*

Now, we present some results to understand why events might be important to be taken into account.

Theorem 4 *Given three TA, A_i $(i = 1, 2)$ and O, and a labeling function $\mathcal{P}$: Props $\mapsto$ $2^{Locs(O)}$. If $A_1 \simeq_{Labels(O)} A_2$ then $A_1 \parallel O \simeq^{\mathcal{P}_1, \mathcal{P}_2} A_2 \parallel O$ where $\mathcal{P}_i$ $(i = 1, 2)$ are the natural extensions of $\mathcal{P}$ to the respective Cartesian products.*

Proof 8 *If B is the continuos event bisimulation relating the initial states of A_1 and A_2, we can extend it to a continuos observational bisimulation between the states of $A_1 \parallel O$ and $A_2 \parallel O$. Simply, $(q, q') \in B^*$ iff $(\Pi_{A_1}(q), \Pi_{A_2}(q')) \in B$ and $\Pi_O(q) = \Pi_O(q')$. Due to the fact that B respects $Labels(O)$ the τ-observational transitions in G_{A_i} wrt. $Labels(O)$ do not synchronize with O, it can be easily seen that the relation we propose is indeed a bisimulation. Moreover, B^* is such that for any $(q, q') \in B^*$ then $[q]_{\mathcal{P}_1} = [q']_{\mathcal{P}_2}$ (since the O projection is the same by definition). Then, by theorem 3, we have that they satisfy the same TCTL formulae.*

By using the previous result, we conclude.

Corollary 3 *Given three TA A_i $(i = 1, 2)$ and O, and a labeling function $\mathcal{P} : Props \mapsto 2^{Locs(O)}$. Assume that $A_1 \simeq_{Labels(O)} A_2$ then $A_1 \parallel O \models_{\mathcal{P}_1} \phi \iff A_2 \parallel O \models_{\mathcal{P}_2} \phi$ where $\mathcal{P}_i$ $(i = 1, 2)$ are the natural extensions of P to the respective Cartesian products.*

3.3 I/O Timed Components

In this section, we define I/O timed components as a timed automaton attached with an I/O interface information that classifies its labels. In Chapter 5, we give semantics to RTS Designs in terms of I/O Timed Components (instead of directly in terms of Timed Automata). We believe that I/O Timed Components are natural models of timed non-blocking and non-zeno behavior (synchronization mechanisms like hand-shakes can be build more realistically on top of these features). There is some work done on preserving reactivity and activity of components. In [26], it is presented an algebraic framework based on the temporal properties of synchronization operation (they aim at getting high level synchronization facilities). Our point of view is a functional classification of transitions, which is suitable for asynchronous modeling. [1] On that line of reasearch, it is worth mentioning [111] where the authors present non-blocking Timed Processes to get a family of automata where they can apply an assume/guarantee reasoning. They do not address non-zenoness. Liveness and I/O interfaces have been considered in a general setting for simulation proof methods "a la" Lynch-Vaandrager [76]. In that work it is defined Live Timed I/O automata using a notion of "responsiveness" based on games which embeds several proposals for fair I/O timed systems [125, 150], etc. Some of our concepts could be theoretically embedded into that framework, but we found more suitable for our framework I/O Timed Components (see bellow).

I/O components is a high-level architectural notion based on TA which has several advantages for our work. Among others:

- it allows us to define when a model is properly built (i.e., non-zenoness). In particular, we provide a set of syntactical constraints to ensure that the parallel composition is

[1] we believe it is possible to integrate both views.

non-zeno.

- It helps to calculate, in a quite precise way, the influence of a component on other component behavior (see Chapters. 6 and 7). As far as we know, this is a completely new goal for I/O interfaces in formal timed systems.
- Since they are build on top of a simple notion of TA "a la" Alur-Dill they are immediately supported by several checking tools like [61, 23]. We do not need to resort to games like in live I/O Timed Automata [76] and sufficient conditions for I/O admissibility are easy to automate.
- we believe that they are more suitable than [111] to model high-level non-blocking abstractions (see discussion at the end of this chapter).
- These notions are independent of the underlying timed (or untimed [2]) formalism used to describe the dynamics.

Let us define these concepts formally:

Definition 17 (Admissible Input/Output interface of TA) *Given a non-zeno TA A, $I_A, O_A \subseteq 2^{Labels(A)}$ (powerset of labels) is an admissible input/output interface for A iff:*

1. *There is a partition [3] P of $Labels(A)$ such that $I_A, O_A \subseteq P$ and $I_A \cap O_A = \emptyset$. Sets $I \in I_A$ and $O \in O_A$ are called input selections and output selections respectively. A label appearing in an input or an output selection is an input label or an output label respectively. The set of input and output labels constitutes the set of interface labels. The labels which are not in the interface labels are internal lables. In other words, we can state the property as (a) no label can be an input and an output label at the same time, and (b) a label belongs to at most one Input or Output selection.*
2. *For any reachable states q and for any input selections $I \in I_A$ there exists a label $i \in I$ and a state q' such that $q \mapsto_0^i q'$. That is, at least one alternative of the selection is enabled.*
3. *For any reachable states there exists a divergent run starting from that state and such that it does not contain any input label.*[4]
4. *For any reachable states q and for any output selections $O \in O_A$, if $q \mapsto_0^o q'$ with $o \in O$ then forall $o' \in O$ there exists a state q'' such that $q \mapsto_0^{o'} q''$. (i.e., all output labels of an "output selection" are available simultaneously)*

[2] We believe that this I/O model can be adapted to the untimed framework by changing timed divergence conditions with fairness constraints [56, 76], which are an usual way to specify progress in the untimed framework.

[3] which includes the empty set

[4] This property is stronger than non-zenoness since we are requiring that non-zenoness does not depend on input labeled transitions. It is similar to Progressiveness of [143] and feasibility of [150]

5. *If a run contains an infinite number of occurrences of a transition which is not an Input-labeled transition, then that run is necessarily time-divergent (non-transientness of outputs and internal transitions* [5]*).*

Note that naturally an input can not block an output (at most, it can define which output of an output selection is enabled). Also, an input is not mandatory and that is why non-zenoness must be guaranteed without them. The non-transientness of output and internal transitions has a more technical motivation. It turns out to be a necessary condition to guarantee that the composition of I/O timed components is also an I/O timed component (see next definitions). In Sect. 3.3.1 we show how to check I/O admissibility.

Perhaps, I/O selections is the concept that needs more rationale. A component providing a set of labels as an input selection I is guaranteeing that at least one transition with label in I is enabled at each reachable state. A component providing a set of labels as an output selection is guaranteeing that whenever there is a transition enabled with a label of the group there are transition enabled for all the rest of the labels. Historically, process algebra [126] featured some kind of blocking synchronization to communicate agents. Usually, they were used to either model a hand-shake like communication between active agents (like ADA or CSP [94]) or to communicate the state of some agent (external choice). We do not want a blocking synchronization as primitive feature (we can build on top of this asynchronous I/O mechanism) but we still want a simple means to communicate events or states. That is why, the Output selection models non-deterministic choices that are finally selected by the input enabled of the corresponding Input selection. That feature is not present in previous I/O models like [76, 111, 150], etc.

Definition 18 (I/O Timed Component) *An I/O Timed Component (or just component) is a tuple* $(A, (I_A, O_A))$ *where* A *is a timed automaton and* (I_A, O_A) *is an admissible I/O interface for* A.

Given an I/O timed component $C = (A, (I_A, O_A))$, $[C]$ will denote its underlying timed automaton A.

Definition 19 (Compatible Components) *Given two components* $C_1 = (A_1, (I_1, O_1))$ *and* $C_2 = (A_2, (I_2, O_2))$*, they are compatible if and only if :*

1. *all the labels of* $Labels(A_1) \cap Labels(A_2)$ *are interface in both* C_1 *and* C_2 *(i.e.,* $Labels(A_1) \cap Labels(A_2) \subseteq (I_1 \cup O_1) \cap (I_2 \cup O_2)$*),*
2. *for all* $I \in I_1$ *and* $I' \in I_2$ *then if* $I \cap I' \neq \emptyset$ *then either* $\#I = 1$ *or* $\#I' = 1$.
3. *for all* $O \in O_1$,$O' \in O_2$ *then* $O \cap O' = \emptyset$.

[5] This requirement together with the previous divergence property and non-zenonnes of the underlying TA are closely related to the notion of Strong I/O Feasibility of [150]

4. for all $O \in O_1 \cup O_2$ and $I \in I_1 \cup I_2$ then $I \cap O = \emptyset$ or $I \subseteq O$.

We refer to a set of pair-wise compatible components as a *compatible set of components.* I/O compatibility means that the underlying TA can not block each other and moreover, we will show that the composition of compatible components is itself a component (Sect. 3.3.1).

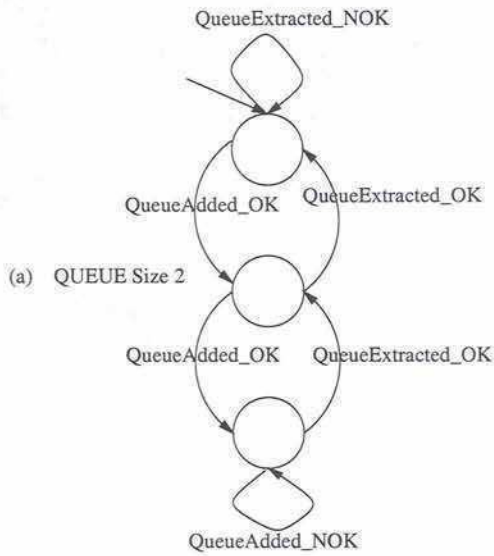
Example: 6 *In Fig. 3.2 we show two compatible I/O components: a 2-size queue and a writer of that queue. Note how the input labels of the queue are enabled or disabled according to the state of the queue (empty, full).*

Example: 7 *In Fig. 3.3 the reader can see a version of the Rail-Crossing System where the control part waits the gate going down before requiring it to go up. This version is for 2 trains but it can parametrically extended to any number of trains just keeping the number of trains in the dangerous zone.*

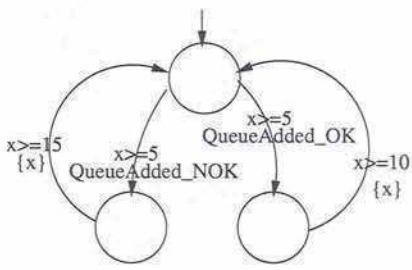
Example: 8 *CSMA/CD (Carrier Sense, Multiple Access with Collision Detection) is widely used protocol on LANs on the MAC sublayer. It solves the problem of sharing a single channel in a broadcast network (a multi-access channel). When a station has data to send it first listens to the channel to check whether it is idle or busy. If the bus seems idle it begins sending the message, else it waits a random amount of time and then repeats the sensing operation. When a collision occurs, the transmission is aborted simultaneously in all the stations that were transmitting and they wait a random time to start all over again. We formally model the timing aspects of the protocol using I/O timed components (see Fig.3.4) based on the model presented in [128]. Sender components share a bus component. We suppose that the bus is a 10Mbps Ethernet with worst case propagation delay σ of 26 ms. Messages have a fixed length of 1024 bytes, and so the time λ to send a complete messages, including the propagation delay, is 808 ms. The bus is error-free, no buffering of incoming messages is allowed. Note that $SendOK_i$ and $SendBusy_i$ is an output selection of the sender and the selection depended on the input actually enabled in the bus state. In fact, $SendBusy_i$ is enabled when the head of a message has already propagated. It takes at most σ to propagate the collision signal to all the senders. The sender stays at most δ in the transmission location. Note also that the sender non-deterministically makes a new attempt to send before 2σ elapsed since the last attempt.*

3.3.1 I/O Components, Composition and Non-Zenoness

Let us state some results that help to prove that a TA-model is non-zeno. Firstly, we will see how an admissible interface can be derived for the parallel composition of two compatible I/O components. This is a rather strong result which encompass the following fact: given two compatible components A_1, A_2 then the composition, which turns out to be non-blocking, is also a component (i.e., $[A_1] \parallel [A_2]$ is non-zeno and moreover it can be given an admissible I/O interface). Briefly, the new input interface is constituted by the

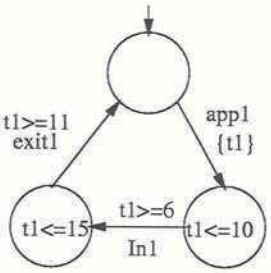


I={{QueueAdded_OK, QueueAdded_NOK}, {QueueExtracted_OK, QueueExtracted_NOK}}
O={{}}



I={{}}
O={{QueueAdded_OK, QueueAdded_NOK}}

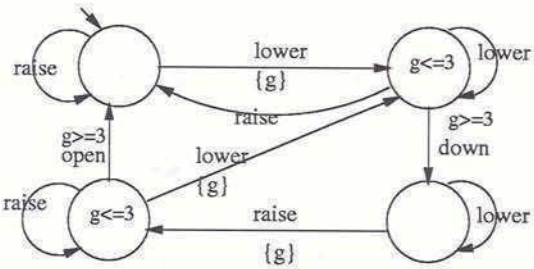
Figure 3.2: Two compaible I/O components



TRAIN 1

I={}

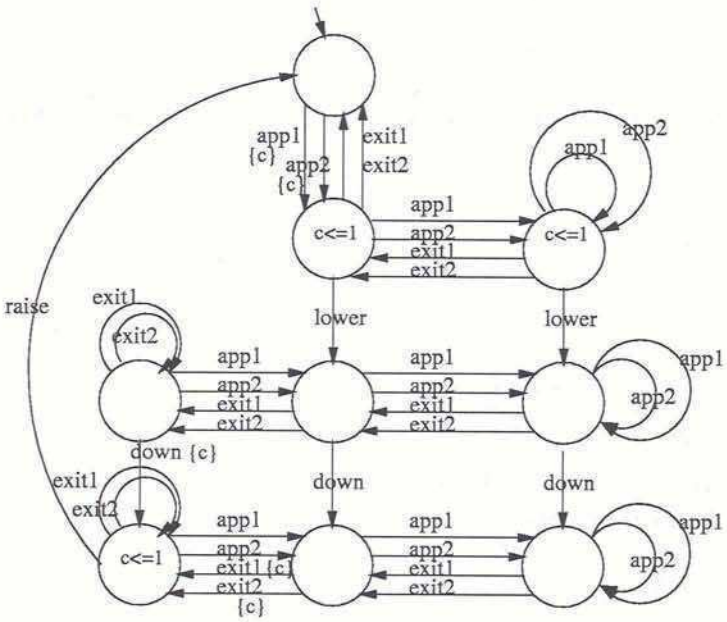
O={{app1},{in1},{exit1}}



GATE

I={{lower},{raise}}

O={{down},{open}}



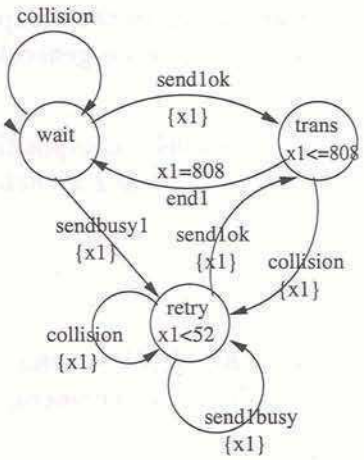
CONTROLLER

I={{app1},{app2},{down},{exit1},{exit2}}

O={{lower},{raise}}

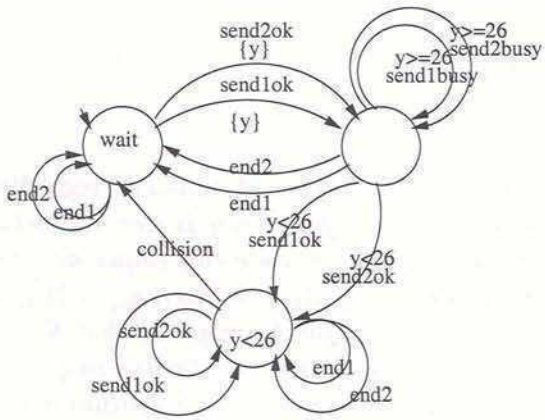
Figure 3.3: I/O Components of the RCS

SENDER 1



I={{collision}}
O={{send1ok,send1busy}, {end1}}

BUS



I={{send1ok,send1busy},{end1},{end2}, { send2ok,send2busy}}}
O={{collision}}

Figure 3.4: I/O Components of the CSMA/CD Protocol

union of the original input selections which at most synchronizes with a singleton input selection, (which is indeed non-blocking thus preserving the "selectivity property" -point 2 - of any input selection containing the singleton). Something similar can be done to build the new output interface. Since output selections that intersect with input selections of size greater than one may loose the simultaneous availability property (point 4 of the definition of I/O), they are not part of the new output selections. However, it can also be added as output selections of size 1 all the labels of those lost output selections (they trivially satisfy point 4). Note that due to the items of compatibility (in particular the second item) all interface labels of the components are promoted to interface labels of the composition component. This fact is important to prove that this construction can be generalized to the parallel composition of n components (see Lemma 4).

In the example 6 the resulting interface of the parallel composition is I= {}, O ={$\{QueueAdded_{OK}\}$, $\{QueueAdded_{NOK}\}$, $\{QueueExtracted_{OK}\}$, $\{QueueExtracted_{NOK}\}$ }.

Formally,

Lemma 5 *Given two I/O-compatible components* $C_1 = (A_1, (I_1, O_1))$ *and* $C_2 = (A_2, (I_2, O_2))$ *then* $C_{1,2} = (A_1 \parallel A_2, I^C, O^C)$ *is an I/o Timed Component, where*

$$\begin{aligned} I^C = & \{I \in I_1 / \nexists I' \in I_2 : I \subset I' \wedge \nexists O \in O_2 : I \cap O \neq \emptyset\} \\ & \cup \{I \in I_2 / \nexists I' \in I_1 : I \subset I' \wedge \nexists O \in O_1 : I \cap O \neq \emptyset\} \end{aligned}$$

and

$$\begin{aligned} O^C = & \{O \in O_1 / \nexists I' \in I_2 : \#I' > 1 \wedge I' \subseteq O\} \\ & \cup \{\{a\} / a \in O \in O_1 \wedge \exists I' \in I_2 : \#I' > 1 \wedge I' \cap O \neq \emptyset\} \\ & \cup \{O \in O_2 / \nexists I' \in I_1 : \#I' > 1 \wedge I' \subseteq O\} \\ & \cup \{\{a\} / a \in O \in O_2 \wedge \exists I' \in I_1 : \#I' > 1 \wedge I' \cap O \neq \emptyset\} \end{aligned}$$

Proof 9 *The most difficult point is the proof that* $A_1 \parallel A_2$ *is indeed non-zeno regardless input transitions: We will see that any reachable state by a finite run is not a timelock (i.e., that finite runs is the prefix of a divergent run). Moreover, time can elapse avoiding input transitions. Let* q *be a reachable state by a finite run of* $A_1 \parallel A_2$ *then* $q_{a_1} = \Pi_{A_1}(q)$ *and* $q_{a_2} = \Pi_{A_2}(q)$ *are reachable states (by finite runs) of* A_1 *and* A_2 *resp. Let* $k \in \Re_{\geq 0}$ *be a constant. From the definition of component, there must be runs* r_1, r_2 *starting in* q_{a_1} *and* q_{a_2} *resp. of time length equal to* k *such that* r_1 *does not contain any* I_1 *transition and* r_2 *does not contain any* I_2 *transition (thus they do not contain any label in* $I_{C_{1,2}}$*). Now, we show a procedure to obtain a run of* $A_1 \parallel A_2$ *from* r_1 *and* r_2*. To obtain such a run, we would need to merge* r_1 *and* r_2*. If the discrete transitions of* r_1 *and* r_2 *are sorted according to the time of occurrence, it is easy to combine them and build a run of* $A_1 \parallel A_2$ *till the first output-labeled transition which must be shared by the other automaton is found. To outline the merge, lets* $r_1 = q_{a_1} \mapsto^{l_1}_{t_1} q_1 \mapsto^{l_2}_{t_2} ...q_n$*, and* $r_2 = q_{a_2} \mapsto^{l'_1}_{t'_1} q'_1 \mapsto^{l'_2}_{t'_2} ...q'_{n'}$*. Now, suppose that* $t_1 \leq t'_1$ *(the other case is symmetrical) and* l_1 *is not shared by* A_2 *(or it is* λ*). Then - thanks to the parallel composition interleaving semantics - the resulting run* $r_{1,2}$ *can be build*

as follows: $r_{1,2} = q \mapsto^{l_1}_{t_1} (q_1, q'_1 + t_1)$ *concatenated with the run obtained using the same procedure from* $(q_1, q_{a_2} + t_1)$ *with* $r_1 = q_1 \mapsto^{l_2}_{t_2} ...$, *and* $r_2 = q_{a_2} + t_1 \mapsto^{l'_1}_{t'_1 - t_1} q'_1 \mapsto^{l'_2}_{t'_2} ...q'_{n'}$. *Clearly this procedure can be iterated finitely till we reach the end of both runs (the variant is sum of the number of transitions of both runs), thus obtaining a run of* $A_1 \parallel A_2$ *of time length* k, *or till a shared label is found.*[6]

Without loss of generality, let us suppose that the earliest still non-synchronized shared output-transition $q_i \mapsto^o_0 q_{i+1}$ *belongs to* r_1 *and* $o \in O \in O_1$. *Let* $I \in I_2$, $I \subseteq O$ *the corresponding matching input selection (i.e.,* $o \in I$*). By item 1, 3 and 4 of compatibility that label belongs to an input selection* I *such that* $I \subseteq O$. *By definition of input selection, there is a transition labeled* $i' \in I$ *enabled in* A_2 *at the time of occurrence of that* i^{th} *transition. By definition of output selection, at* q_i *there must be also a discrete transition* $q_i \mapsto^{i'}_0 s$. *By applying this procedure, we can fix up both runs to get a finite run starting at* q *such that either it has time length* k *or it ends with an output transition into an intermediate state* q'. *Therefore, since both TA are non-transient for output labeled transitions (item 5 of I/O interface admissibility), by repeating the whole procedure from those intermediate states (i.e., obtained new* r_1, r_2, *etc.), a run of time length* k *is eventually built (if not, either the projection of that infinite run on* A_1 *or* A_2 *would show an infinite number of output-labeled transitions, and since there is a finite number of labels at lest one output label would be repeated infinitely often thus violating the last item of I/O interface admissibility) Let's see the rest of the items of I/O interface:*

- *the new input and output labels are disjoint (input selections intersecting with an output selections are not part of the new interface).*
- *Input selection property: given an state* q *of* $A_1 \parallel A_2$ *and an input selection* I *of* $I_{C_{1,2}}$, *we know that* I *belongs either to* I_1 *or to* I_2. *Without loose of generality, lets suppose that it belongs to* I_1. *Then, there exists* $i \in I$ *such that* $\Pi_1(q) \mapsto^i_0 r$. *We also know that if* $i \in Labels(A_2)$ *then* $\{i\} \in I_2$ *(input selection of size 1) and thus there exists* s *such that* $\Pi_2(q) \mapsto^i_0 s$ *and then* $q \mapsto^i_0 (r, s)$.
- *Output Selection Property: Similar to the previous one.*
- *finally, a run containing an infinite number of internal or output-labeled transitions is necessarily time-divergent. Indeed, since any run of* $A_1 \parallel A_2$ *can projected into a run of* A_1 *and a run of* A_2 *and one of those runs must exhibit an infinite number of outputs or internal transitions and therefore diverge.*

Corollary 4 *Given an indexed set of* n *components* S *such that they are pair-wise compatible then* $(\|_{1 \leq i \leq n} A_i, (I^n, O^n)$ *is a component where*

$I^n = \bigcup_{1 \leq i \leq n} \{I \in I_i / \nexists k \neq i, I' \in I_k : I \subset I' \wedge \nexists k \neq i, O \in O_k : I \cap O \neq \emptyset\}$ *and,*

[6] Note that if one of the runs is empty then it just remains a set of discrete (0 time) transitions in the other run (both have originally the same time length) and therefore we can omit that suffix since we have already built a run of time length k.

$O^n = \bigcup_{1 \leq i \leq n} \{O \in O_i / \nexists k \neq i, I' \in I_k : \#I' > 1 \wedge I' \cap O \neq \emptyset\} \cup \{\{a\}/a \in O \in O_2 \wedge \exists k \neq i, I' \in I_k : \#I' > 1 \wedge I' \cap O \neq \emptyset\}$.

Proof 10 *By induction. Base case is solved by the last theorem. Case n+1. By inductive hypothesis we know that*

$(I^n, O^n) = (\bigcup_{1 \leq i \leq n} \{I \in I_i / \nexists k \neq i, I' \in I_k : I \subset I' \wedge \nexists k \neq i, O \in O_k : I \cap O \neq \emptyset\}, \bigcup_{1 \leq i \leq n} \{O \in O_i / \nexists k \neq i, I' \in I_k : \#I' > 1 \wedge I' \cap O \neq \emptyset\} \cup \{\{a\}/a \in O \in O_2 \wedge \exists k \neq i, I' \in I_k : \#I' > 1 \wedge I' \cap O \neq \emptyset\}))$

is an admissible I/O interface for $\|_{1 \leq i \leq n} A_i$. Let us call that component C^n. We know that $C_{n+1} = (A_{n+1}, (I_{n+1}, O_{n+1}))$ is compatible with all $C_i = (A_i, (I_i, O_i))$ $1 \leq i \leq n$. Let's see that is compatible with the interface for the n components but firstly lets pinpoint some facts about the interface (I^n, O^n) of C^n.

1. *An interface label of C_i $(1 \leq i \leq n)$ is an interface label of C^n. This comes from the following facts: (a) Input labels remain as input labels in the biggest input selection containing it except in the case that the input selection matches with an output selection (in that case, $I \subseteq O$)., and (b) Output labels remain in the interface.*

2. *Input selections of I^n are input selections of some of its constituent components (i.e., If $I \in I^n$ then there exists $k \in \mathbb{N} : 1 \leq k \leq n$ such that $I \in I_k$)*

3. *If O is an output selection of O^n, then there exist $k \in \mathbb{N} : 1 \leq k \leq n$ such that $O \in O_k$ for some $k : 1 \leq k \leq n$ or there exists $O' \in O_k$ and $O = \{a\} \subseteq O'$, and there exists $m : 1 \leq m \leq n$ such that $I' \in I_m$ and $I' \subseteq O'$.*

Therefore, suppose that A_{n+1} has a common label with $\|_{1 \leq i \leq n} A_i$ then, for instance, that label belongs to a k^{th} automata and therefore belongs to the interface of the components C_k and C_{n+1}. If that label is an output label of the C_{n+1} component, that label still belongs to the interface C^n due to the first observation.

The second compatibility item ($I \cap I' \neq \emptyset$ then either $\#I = 1$ or $\#I' = 1$) is trivially true due to the observation that input selections of C^n are input selections of the original components and the pairwise compatibility. Similarly, if an output selection O of O_{n+1} intersects with some input selection I of I^n then that input selection must be an input selection of some component and therefore that input selection must be included in the output selection (i.e., $I \subseteq O$). If an input selection of I of I_{n+1} intersects with some output selection of O^n then either it is an input selection of size 1 and and it is trivially included, or, by the last observation, we know that there exists $O \in O_k$ and $O \in O^n$ and thus $I \subseteq O$ (that is due to pairwise compatibility, I must be the only input selection of size greater than 1 intersecting with O and then by the last observation O must belong to O^n).

Therefore, they are compatible components and by Lemma 5:

$(I'^{n+1}, O'^{n+1}) =$

$$I'^{n+1} = \{I \in I^n / \nexists I' \in I_{n+1} : I \subset I' \wedge \nexists O \in O_{n+1} : I \subseteq O\} \cup \{I \in I_{n+1} / \nexists I' \in I^n : I \subset I' \wedge \nexists O \in O^n : I \cap O \neq \emptyset\}$$

$$O'^{n+1} = \{O \in O^n / \nexists I' \in I_{n+1} : \#I' > 1 \wedge I' \cap O \neq \emptyset\} \cup \{\{a\}/a \in O \in O_1 \wedge \exists I' \in I_{n+1} : \#I' > 1 \wedge I' \cap O \neq \emptyset\} \cup \{O \in O_{n+1} / \nexists I' \in I^n : \#I' > 1 \wedge I' \cap O \neq \emptyset\} \cup \{\{a\}/a \in O \in O_i \wedge \exists I' \in I^n : \#I' > 1 \wedge I' \cap O \neq \emptyset\}$$

It is not difficult to see that this interface is equivalent to: (I^{n+1}, O^{n+1}) *where*

$$I^{n+1} = \bigcup_{1 \leq i \leq n+1} \{I \in I_i / \nexists k \neq i, I' \in I_k : I \subset I' \wedge \nexists k \neq i, O \in O_k : I \cap O \neq \emptyset\} \text{ and}$$

$$O^{n+1} = \bigcup_{1 \leq i \leq n+1} \{O \in O_i / \nexists k \neq i, I' \in I_k : \#I' > 1 \wedge I' \cap O \neq \emptyset\} \cup \{\{\overline{a}\}/a \in O \in O_2 \wedge \exists k \neq i, I' \in I_k : \#I' > 1 \wedge I' \cap O \neq \emptyset\}$$

In fact, if we write I^{n+1} *in terms of* I^n *we need to add the input selections of* I_{n+1} *which are not strictly included in an Input Selection of other* I_k *and do not match with an output selection. On the other hand, we have to eliminate from* I^n *the input selections strictly included in an input selection of* I_{n+1} *and the ones that match with an Output Selection of* O_{n+1}*. That is,* $I^{n+1} = (I^n - \bigcup_{1 \leq i \leq n} \{I \in I_i / \exists I' \in I_{n+1} : I \subset I' \vee \exists O \in O_{n+1} : I \cap O \neq \emptyset\}) \cup \{I \in I_{n+1} / \nexists k \neq i, I' \in I_k : I \subset I' \wedge \nexists k \neq i, O \in O_k : I \subseteq O\}$

Lets see that the definition of I'^{n+1} *specifies that manipulation: Note that, though* I^n *may contain less Input Selections than the union of them* $(\bigcup_{1 \leq i \leq n} I_i)$*, it is easy to see that (a) If an input selection of the union is not present in* I^n *then either it is included on another input selection of* I^n*, or it intersects an output selection of* O^n*, and (b)* $\exists O \in O^n : I \cap O \neq \emptyset$ *iff* $\exists k \leq n, O \in O_k : I \cap O \neq \emptyset$ *(all output label remains). Therefore, the set* $\{I \in I_{n+1} / \ / \exists k \neq i, I' \in I_k : I \subset I' \wedge \nexists k \neq i, O \in O_k : I \subseteq O\}$ *is equivalent to* $\{I \in I_{n+1} / \nexists I' \in I^n : I \subset I' \wedge \nexists O \in O^n : I \cap O \neq \emptyset\}$*. This proves that in* I'^{n+1}*, the same input selections of* I_{n+1} *filtered by the* I^{n+1} *are present. Finally,* $I^n - \bigcup_{1 \leq i \leq n} \{I \in I_i / \exists I' \in I_{n+1} : I \subset I' \vee \exists O \in O_{n+1} : I \cap O \neq \emptyset\}$ *is equivalent to* $\{I \in I^n / \nexists I' \in I_{n+1} : I \subset I' \wedge \nexists O \in O_{n+1} : I \subseteq O\}$ *and we can conclude that* $I'^{n+1} = I^{n+1}$*.*

On the other hand, to write O^{n+1} *in terms of* O^n*, the output selections of* O_{n+1} *that do not match with input selections of size greater than one must be added as well as the singletons for the ones that match. Besides, the output selections of* O^n *must be checked against the input selections of* I_{n+1} *to eliminate and convert into singleton output selections the ones that match with input selections of size greater than one. Again, this is specified by the definition of* O'^{n+1}*.*

Guaranteeing I/O Admissibility

For the sake of self containment we provide sufficient syntactic constraints and checking-algorithms to guarantee that $(A, (I_A, O_A))$ is indeed a component.

For example, to satisfy the property of input being non-blocking (2nd condition), we

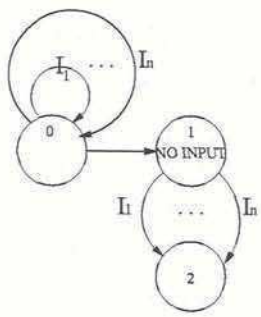


Figure 3.5: Observer for Checking Non Zeno Regardless Input

can resort to the following syntactic property: $\forall I' \in I_A : \forall l \in Locs(A) : I(l) = \bigvee_{\{e:Label(e)\in I' \wedge src(e)=l\}} Guard(e)$. That is, while the invariant is valid at least one I'-labled transition is labeled.

To check that any output selection is simultaneously enabled (4th condition) one of the possible syntactic property is the following: $\forall l \in Locs(A), \forall O \in O_A : (\exists e \in Edges(A) : src(e) \in l \wedge Label(e) \in O) \Rightarrow (\forall o \in O \exists e' \in Edges(e) : src(e) = l \wedge Label(e') = o \wedge Guard(e') = Guard(e))$. We are asking that we can find a set of edges with the same guards for the output selection (the property is enogh for our purposses).

To check non-zenoness we use an observer automaton with three locations: Location 1 is entered non-deterministically from initial location 0 and it is left to go to a trap location 2 whenever input occurs. Then, we ask whether $A \parallel Observer$ satisfies the following TCTL [92] formula : $\forall\Box @ = 1 \rightarrow \exists \diamond_{\geq 1} @ = 1$, i.e., whether time can elapse without traversing an input edge (See Fig. 3.5).

For non-transientness of outputs, it suffices to require that no pair of outputs or internal events can occur closer than one time unit. This can be done by resorting to an observer TA or, alternatively, adding and checking some syntactic constraints on output and internal edges, for instance, having a minimum delay guard on a clock reset in the potential previous events.

Discussion: I/O Components vs Non-Blocking Timed Processes

Like our work, in [111] the notion of non-blocking I/O Timed Processes is introduced to deal with non-blocking avoiding games (like in [76]). Their approach is different from ours in subtle and mayor topics. On the one hand, I/O Timed Processes communicate themselves by "listening" the change of output mapping of their partners. Output mappings are associated with each location. Informally, to be non-blocking a process should satisfy two conditions: (a) a positive real amount of time can elapse (with an optional change of output mapping) (b) that can happen regardless the input change. That is, while the update of outputs is independent of the update of inputs, the update of internal variables (i.e., the location and the clocks) depends on it. Those rules are closely related to our non-blocking

requirement for inputs and, partially, to the non-zenoness regardless inputs. However, in their setting there is no necessity of requiring non-zenoness.

On the other hand, their communication approach is suitable for modeling circuits but it could become cumbersome for modeling high-level software abstractions. For example, to model a queue in the Timed Process setting, the reader process should receive queue state changes (even the ones not performed by itself, thus a fictious communication) to keep track of the queue state to output a successful or unsuccessful reading (if the reader process performs different changes of output depending on the queue output it would be violating condition (b)). That greatly complicates the readers/writers models. As seen in the examples, we can resort to I/O selections to perform different outputs according to the partner state. As far as we know, that modeling feature is not present in any other I/O proposal.

Chapter 4

Describing RTS-Designs and Requirements

When developing a RTS, designers usually should deal with four different descriptions:

1. the system architecture;
2. the internal dynamics of tasks;
3. the behavior of communicating components and the environment where the system is embedded; and finally
4. the requirements or properties the system must satisfy.

We have considered analyzability of resulting designs as the main criterion for defining the features of the design notation. In particular, our technique is based on the use of known analytical scheduling theory for Fixed-Priority scheduling [109] to build a formal model. Thus, theory assumptions to calculate WCCT and the expressive power of selected target formalism set some restrictions for language features (e.g., there is no dynamic task creation, tasks are not migratable, tasks do not suspend themselves, etc.). Although we believe that some limitations can be overcome, there is a trade-off between sophistication of supported features and the ability to efficiently predict the behavior of the resulting designs. In the next sections, we present examples of such notations while we point out some of the essential issues which constrain their style and features.

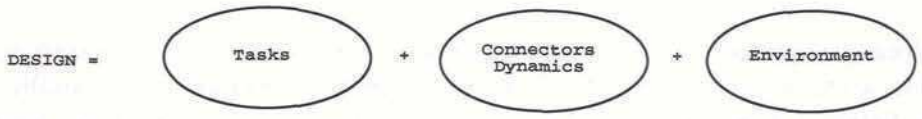


Figure 4.1: Design Elements

4.0.2 Introduction to Fixed Priority Application Model

In this section, we outline the main features of the application model we adhere to. Our application model is strongly based on the Fixed-Priority application models (e.g.,[109]). It assumes a fixed set of tasks on each processor scheduled under Fixed-Priority policy. A base priority is assigned to them. A task execution (a job) can be preempted by any job of a task of higher priority. Jobs do not suspend themselves while executing.

In these applications, two kinds of tasks coexist: periodic and sporadic tasks. For the periodic case, we require each job of a task to complete its execution within a task period. Sporadic tasks respond to signals (trigger events) which have a minimum separation assumption (minimum intearrival time) through a sporadic server scheme [144, 109]. In particular, our instantiation of the scheme works as follows: if a signal arrives and more than the declared minimum interarrival time has elapsed since the last server job release then a new job is released. Otherwise, the signal is lost or latched according to the designer decision. We require a job completes in less than the minimum interarrival time. Then, sporadic tasks can be treated as periodic ones for the analysis (for us, the minimum separation time plays the role of the period).

To achieve mutual exclusion while accessing to data, the applications use an emulation of the Priority Ceiling Protocol [109, 141]. PCP emulation scheme requires the critical section (monitor server code) to execute at a level slightly higher than the priority of any client task that may try to perform an operation on the shared resource [86]. PCP emulation guarantees a bounded blocking time.

The time required to perform scheduling, context switching and other overheads is ignored. Furthermore, to simplify the presentation of this notation, there are no explicit features to describe software mode change, jitter on clocks, phase offsets or suspension.

To assess schedulability under the assumptions of a fixed-priority model, a worst case completion calculus is usually provided (see for example [86]). WCCT calculus is an analytical tool for computing the worst case response times of task or subtask measured from release times. Roughly speaking, the WCCT (or response time) for a task τ_i can be calculated as the least fixed point of an equation (see Chapter 5). That equation takes into account the worst-case interference of tasks of higher or equal priority plus the blocking time due to lower priority tasks accessing to mutual exclusion areas. In Sect. 5.3, we present the details of the WCCT calculus for our fixed-priority application model.

4.1 Describing the System Architecture

As already explained, we adapted elements of periodic application model to describe execution architectures. Thus, detailed designs may be composed of periodic and sporadic tasks distributed in a network of processors (see Fig. 1.2). Periodic tasks are released periodically while sporadic tasks are released by the arrival of a triggering event. The language

also supports the definition of protected objects to model data and control communication among tasks. As we will see later in this chapter, bounded queues, signal nets, circular buffers and timers can be defined. Moreover, most finite and non-blocking abstractions could be easily incorporated into our proposal. That is, since the version of the WCCT calculus we present does not deal with unbounded suspension, we limit blocking only to achieve mutual exclusion (and an error is returned if the action cannot be performed due to "semantical" reasons; for example when writing to a full queue, etc.).

Processing nodes may communicate through a signal net in an asynchronous fashion (i.e., a sporadic server attends the signal). On the other hand, the controller and the plant communicate by means of data written in and out at ports or signaling mechanism which triggers sporadic servers.

As was shown in the introduction, we use a simple graphical notation based on HRT-HOOD [42] to define a net of processors, and the distribution and connection of the components.

4.2 Task Dynamics

Task behavior is specified at a level of abstraction where only the communication patterns and the estimates on the execution times of actions are described - neither data nor functionality are specified.[1]

In this chapter we deal with three different language levels to describe task behaviors: Design Language, Basic Terms and CDAGs. In few words, CDAGs are annotated control DAGs while Basic Terms is a structured language translatable into CDAGs. Design Language is just a higher-level version of Basic Terms featuring macro-expansion mechanisms (e.g., it allows the designer to redistribute task code into protected objects)

4.2.1 A Structured Language to Define Task Dynamics: Design Language

Now we introduce a structured language featuring some syntactic sugar to ease the description the abstract code performed by tasks.

Basic Terms

Abstract code is described using a very simple sequential external-internal action language to model asynchronous processes. The *terms*, which are the core of the language, are

[1]The reader can see later in this chapter that finite state data domains could be represented and manipulated. However, we suggest avoiding modeling data and functionality whenever those aspects seem irrelevant for the satisfaction of timing requirements.

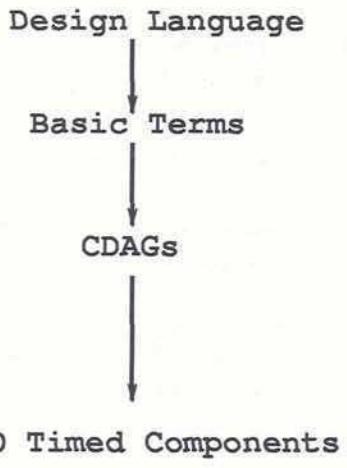


Figure 4.2: Language Levels for Describing Tasks Dynamics

defined by a recursive type definition:

$$T \;=\; action; T \mid action; Set[branch]; T \mid \epsilon$$

where

$action =< [Min, Max], Pty, finalevents >$ where Min, Max are rational numbers, Pty is a natural number, $finalevents$ is a set of strings; and $branch =< EventConditions : Set[strings], Code : T >$

Note that a branch set is always preceded by an action. To be a valid term, the following properties must hold:

- $Min \leq Max (where Max > 0)$ are the minimum and maximum computation requirements of the action.
- The priority of the first action is called the base priority and it is the minimum priority of all actions appearing in the term. The last actions of a term should have that base priority (this is a reasonable assumption to simplify WCCT calculus; note also that IH can have greater priority than the server but this case is treated specially by our method see Sect. 5.3).
- The union of the EventConditions of the branches should be equal to the set of final events of the precedent action.
- An event condition set can be empty only if the set of final events of the previous action is empty.
- Given A, B two actions in a term, if $FinalEvents(A) \cap FinalEvents(B) \neq \emptyset$ then $FinalEvents(A) = FinalEvents(B)$

That is, actions are described as the estimated range for the duration of the internal functionalities when running in a dedicated environment (the minimum and maximum computation requirement measured, for example, in milliseconds). The event names will denote possible endings of that operation. Those endings are, in principle, non deterministically chosen but in Sect. 4.3 we show how designers constrain the event occurrence by means of constraining components. A branch is a set of terms (which will be denoted separated by + symbol). Branches models conditional computation. They are chosen according to the final events event of the precedent action (i.e., any branch such that its event conditions include the final event chosen in the last action).

Design Language

Based on HRT-HOOD [42], our design language uses three concepts: *periodic tasks*, *sporadic tasks* and *protected object.*

For a periodic task, its period, priority and abstract code description must be specified. In the case of the sporadic task, its triggering event, minimal interarrival time, priority and the abstract code must be specified. It is also possible to inform the priority and duration of its interruption handler whenever the sporadic server code is mapped into a different task, which is signaled by this interruption handler. Besides, the designer can specify a latching mechanism to retain disabled interruptions, it must specify when to latch them (no latch, latch while waiting for replenish time, or latch always) and the maximum number of interruptions that can be latched. For the sporadic case, the designer can also specify abstract code to be executed as initialization (see the design for the Active Structure Control System). The designer may also define new protected objects simply as collections of operations: abstract code running at a priority ceiling.

Abstract code are valid terms where no priority is specified since it is the priority of the corresponding task or protected object. To invoke protected object code, a macro expansion mechanism is featured by the abstract code. That is given an abstract code, a basic term may be obtained by marking internal actions with the base priority of the task and by macro-expanding operations on protected objects at their priority ceiling. As another syntactic sugar, event names are translated into the *eventBy*TASKID where TASKID is the respective name of the task performing the code. The goal of this feature is to avoid name conflicts among events of tasks that would be misinterpreted as the same event otherwise (and therefore synchronized!). This feature can be disabled by adding a period at the end, to denote an absolute name for the event.

To illustrate the use of this language, we provide some of the internal descriptions of task and objects for the examples. Note that when Max and Min computation requirement coincide we just write one of them. Also, it is not necessary to write empty sets of final events. Branches are separated by the "+" sign. By default, if not specified, final events of an action preceding a set of branches is the union of condition events of the branches. The whole code can be find in the Annex B.

Example: 9 *Here we describe the dynamics of the sporadic task "Modeler" by using the Design Language. Note that the Model.Update is an invocation (i.e., macro expansion) of the protected object "Model". Also, this example illustrate how initialization for sporadic tasks can be specified.*

```
Modeler (Sporadic, MAT = 100, Pty= 10,
        Interruption=DataReceived, LatchUpTo 1 at WaitingForReplenish)

Init: [10]{ModelerReadyForComm.}
Begin
[18,23]; Model.UpDate; [10] {ModelerReadyForComm.}
End

Model (Protected, PtyCeiling = 12)
Begin
    Operation: UpDate
    Begin
    [2]{ModelUpDated.}
    End

    Operation: Read
    Begin
    [1]{MoldelRead.}
    End
End
```

Example: 10 *This example is taken from the Mine Pump Design. Here, the periodic watchdog tries to extract for the "ACKQ" queue an acknowledgment and there are two course of actions according to the success or not of the operation. The actual dynamics of the queue will be given using constraining components as is shown in the next section.*

```
HLW-WatchDog (Cyclic, Period = 500 ms, Pty= 6)
Begin
[1];AckQ.Extract({AckQExtracted_{OK}.}
                        [1]{ackrequested}
                +
                 {AckQExtracted_{NOK}.}
                        Console.Alarm
                 );[.5]
End

AckQ (Protected, PtyCeiling = 13)
Begin
    Operation: Add
```

```
    Begin
    [2]{AckQAdded_OK, AckQAdded_NOK}
    End

    Operation: Extracted
    Begin
    [2]{AckQExtracted_OK, AckQExtracted_NOK}
    End
End
```

Example: 11 *This is another example taken from the Mine Pump Design. Here, the periodic sensor firstly checks the availability of a result requested to the sensor device in the previous period. If it is not ready, then it annotates in the protected object "Console" the warning situation. If that value is available, it compares it with the value stored in the "CH4Status" object and decides whether it is a safe or unsafe situation. Then it invokes the corresponding action of the "Motor" protected object. The actions are logged and a new value is requested to the CH4 sensor device. In the next section we will show how can be specified the behavior of that device. "Motor", "Logger", and "Console" actions are shown in the complete example (Annex B).*

```
CH4-Sensor (Cyclic, Period = 80 ms, Pty= 10)
Begin
[2]; ({CH4sensorNotReady.}
        Console.Alarm
     +
      {CH4done.}
        [.5]{CH4read};[1];CH4Status.Read;[1];
         ({NotSafe.}
           Motor.NotSafe;CH4Status.Write;
           Console.Alarm
         +
          {Safe.}
           Motor.Safe;CH4Status.Write
          );Logger.Log
      );
[.5]{CH4set.}
End
```

4.2.2 The Kernel Language: CDAGs

At a kernel level source code can be described using directed acyclic graphs with a single root (see Fig. 4.3 and Fig. 4.4) called CDAGs. In next section, we show how to give semantics to CDAGs by means of I/O Timed Components. Non-empty sequences of actions are associated with all non-final nodes. An action is a tuple $action =< [Min, Max], Pty >$

where Min, Max are rational numbers such that $Min \leq Max$ ($Max > 0$) are the minimum and maximum computation requirement respectively (denoted just $[Max]$ when $Min = Max$) and Pty is a natural number which defines the priority of that action. Edges are labeled and they model a possible final event of the associated sequence of actions of its source location (node). Those final events are, in principle, non-deterministically chosen but in the next section we show how event occurrences are constrained.

Definition 20 *Given a node n of a CDAG, let* $O(n) =_{def} \{label(e)/Src(e) = n\}$, *this is the set of final events for the location n.*

There are two conditions that a CDAG should satisfy to be well defined :

- the priority of the first action of a CDAG, called the base priority and denoted ($Pmin$), is the minimum priority of all actions appearing in the DAG. The last actions associated with last nodes should have that base priority (this is a reasonable assumption to simplify WCCT calculus)[2], and
- Given two nodes n, n' of the CDAG such that $O(n) \cap O(n') \neq \emptyset$ then $O(n) = O(n')$. As we will show later, final events can be selected by the environment of the task (e.g., a connector or variable state); intuitively, we require the alternatives to be coherently the same through the CDAG. [3]

In a given design, CDAGs of two tasks should not share any event label. Note that this does not rule out one task executing the triggering event of another task since triggering events are not explicitly annotated in the CDAGs of sporadic tasks (see Sect. 5.1).

As was previously claimed, CDAGs are the kernel language and Design Notation can be translated into them.

4.2.3 From Design Language to CDAGs

As already mentioned, abstract code written using the design language can be translated into basic terms by a simple macroexpansion mechanism. The semantics of basic terms can be described in terms of single-rooted directed acylcic graphs CDAGs using the following procedure:

The term is translated into a single-rooted directed acyclic graph. Its locations are annotated with sequences of actions. This CDAG D satisfies the following properties:

- If p is a path of the control flow of the term, then there is a path in the CDAG D such that p can be obtained concatenating the associated sequences of actions of each path

[2] Note also that, although Interruption Handlers for Sporadic Tasks can have greater priority than the sporadic server, this case is treated specially by our method (see Sect. 5.3).

[3] This property is needed to get an I/O interface for tasks TA (See. Chapter 5).

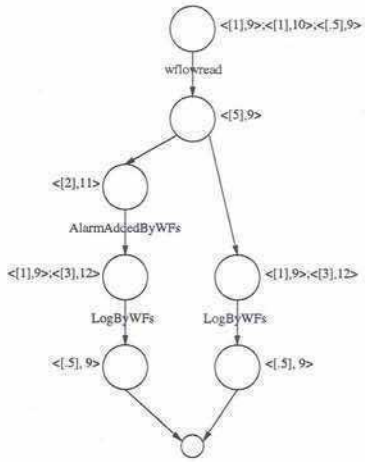


Figure 4.3: CDAG for WaterFlow Sensor

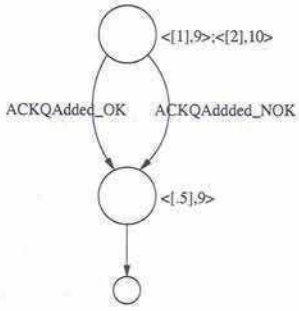


Figure 4.4: CDAG for ACK Handler

location (i.e., $p \in CtrlFlows(Term) \iff p \in \{s_1\&...\&s_n / s_i = Actions(l_i) \wedge l_1...l_n \in Paths(D)\}$) [4]

- The edges leaving a location are labeled with the events associated with the last action of its annotated sequence of actions.

The following algorithm builds such a CDAG according to the degree of accuracy required in the final model (by allowing or not to have sequences of actions not belonging to the control flow of the original term).

$Trans(A)$ is:

Case A

1. $A = \epsilon$ then the result is a graph with a root and a leave locations. The root location annotated with ϵ.
2. $A =< [Min, Max], Pty, E >; T$ where $E \neq \emptyset$: Add a root location annotated $< [Min, Max], Pty, E >$ to $Trans(T)$. For each $e \in E$ add a edge from this new location to the root of $Trans(T)$ labeled e.
3. $A =< [Min, Max], Pty, \emptyset >; T$: Add the action $< [Min, Max], Pty, \emptyset >$ at the head of the sequence associated with the root of $Trans(T)$.
4. $A =< [Min, Max], Pty, E >; (\bigoplus_{1 \leq i \leq n} E_i \ T_i); T$. Build the following DAG: Root location annotated with $< [Min, Max], Pty, E >$ linked to the root of $Trans(T_i; T)$ $(1 \leq i \leq n)$ subDAG with edges labeled e for each $e \in E_i$.

   To obtain a smaller DAG (but loosing accuracy) apply the following procedure: Build a DAG with a root location annotated with $< [Min, Max], Pty, E >$ linked with the $Trans(T_i)$ subDAG $(1 \leq i \leq n)$ by using edges labeled e for each $e \in E_i$. Then, add edge from the leaves of this DAG to the root of $Trans(T)$ labeled according to the final events of their associated sequences.

4.3 Communication and Environment: Constraining Components

In principle, the occurrences of final events of operations and external triggering events for sporadic tasks are not constrained. In this section, we show how to constrain them by means of I/O Timed Components (see Chapter 3) and thus build connectors and environment descriptions. For example, in Fig. 4.5 shows how to model a timer, a non blocking queue -operations fail or succeed according to the queue state-, a perfect net (with latency) and a net which may fail. In particular, if we replace the queue generic event

[4] $CtrlFlows(\epsilon) = \{\epsilon\}$, $CtrlFlows(a; T) = Map(\lambda a.\lambda, CtrlFlows(T))$, $CtrlFlows(a; (\bigcup_{1 \leq i \leq n} b_i); T) = \bigcup_{1 \leq i \leq n} CtrlFlows(a; b_i; T)$.

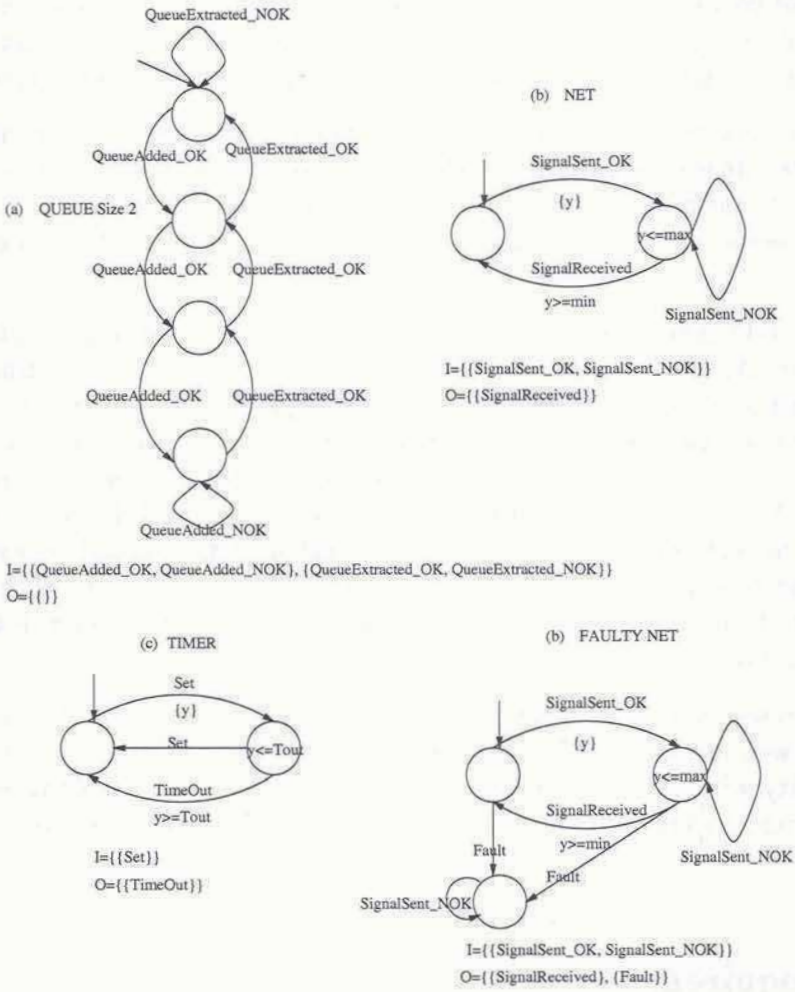


Figure 4.5: Connectors Modeled

names with $AckQAdded_{OK}$, $AckQAdded_{NOK}$, $AckQExtracted_{OK}$, $AckQExtracted_{NOK}$ it becomes a constraining component for the task $ACKQhandler$ of Fig. 4.4.

Assumptions about the environment behavior where the software is embedded may be also modeled by means of Constraining Components (see Fig. 4.6, Fig. 4.7, 4.8, and 4.9). Using I/O Timed Components, designers can specify minimal separations between events, maximum number of events in a period, mode changes, etc. It is also possible to model separately physical events and the arrival of information into actuators and sensors. For instance, it is possible to take into account the time it takes a physical event to be informed by a sensor and the time it takes an actuator to achieve a physical change.

In Fig. 4.6 we can see the model for the actuator, the sensor and a communication component. The sampling takes within 50 and 55 t.u., then the data is prepared within 10 t.u. The actuator applies the pulse for a duration within 25 to 30 t.u., and then data is

prepared for communication (10. t.u. as well). The *Comm* component exemplifies a model of communication; when both parties want to communicate (a handshake) it takes within 3 and 5 t.u. to perform the communication (message and acknowledgement).

Generically, we refer to connectors and environment I/O timed components as *constraining components*. Hence, although the behavior of the environment and the connectors could be given in terms of a more user-friendly language (with some built in templates), for the sake of presentation simplicity and expressive power, we decided to base our notation on TA.

We require I/O compatibility presented in Section 3.3 within tasks and the constraining components. As it can be seen in Chapter 5, the I/O interface for a timed automaton A modeling a task is the following: $I = \{\{triggering\ event\}\}$ if the task is sporadic or $I = \emptyset$ in the periodic case; $O = \{E/\exists l \in Locs(Dag(A)) \wedge E = \{label(e)/src(e) = l\}\}$ (where $Dag(A)$ is the underlying CDAG of the task automaton (see Sect. 5.1)). Two tasks TA are clearly I/O compatible among themselves if their CDAG do not share any label. In a system, correct constraining components must be I/O compatible with the tasks and constraining components they synchronize with. This would mean that given constraining component A and a location l of a task DAG if $O(l) \cap Labels(A) \neq \emptyset$ then the set $O(l)$ must be an input selection of A.

It is easy to see that the components presented for modeling variable states, queues, nets, timers, as well as the components modeling assumptions about the environment, satisfy compatibility with the tasks of the corresponding examples. The designer can define constraining components provided they are I/O compatible with the tasks they synchronize with.

4.4 Requirements

In our proposal, designers ask whether the system may behave in such a way that the analyzed property may be violated. The designer can describe an I/O Timed Component which synchronizes with those events relevant for expressing the requirement. We call those components *Observer Components*.

The labels which are available to write an observer are the events appearing in the descriptions of tasks behavior and the labels that stand for the completion and release of tasks (e.g., *New Period*, *ReplenishTime*, etc.).

An *Observer Component* is an I/O Timed Component compatible with all the components of the *SUA* (*System Under Analysis*). A sufficient condition to build a compatible component with the *SUA* is to build a component where all its shared labels with the *SUA* are input selections of size 1. Those are non-blocking TA "listening" to the events of the *SUA*.[5]

[5] An observer could also be an active component interacting with the *SUA*. However, the designer should be more careful in order to guarantee compatibility with the *SUA*.

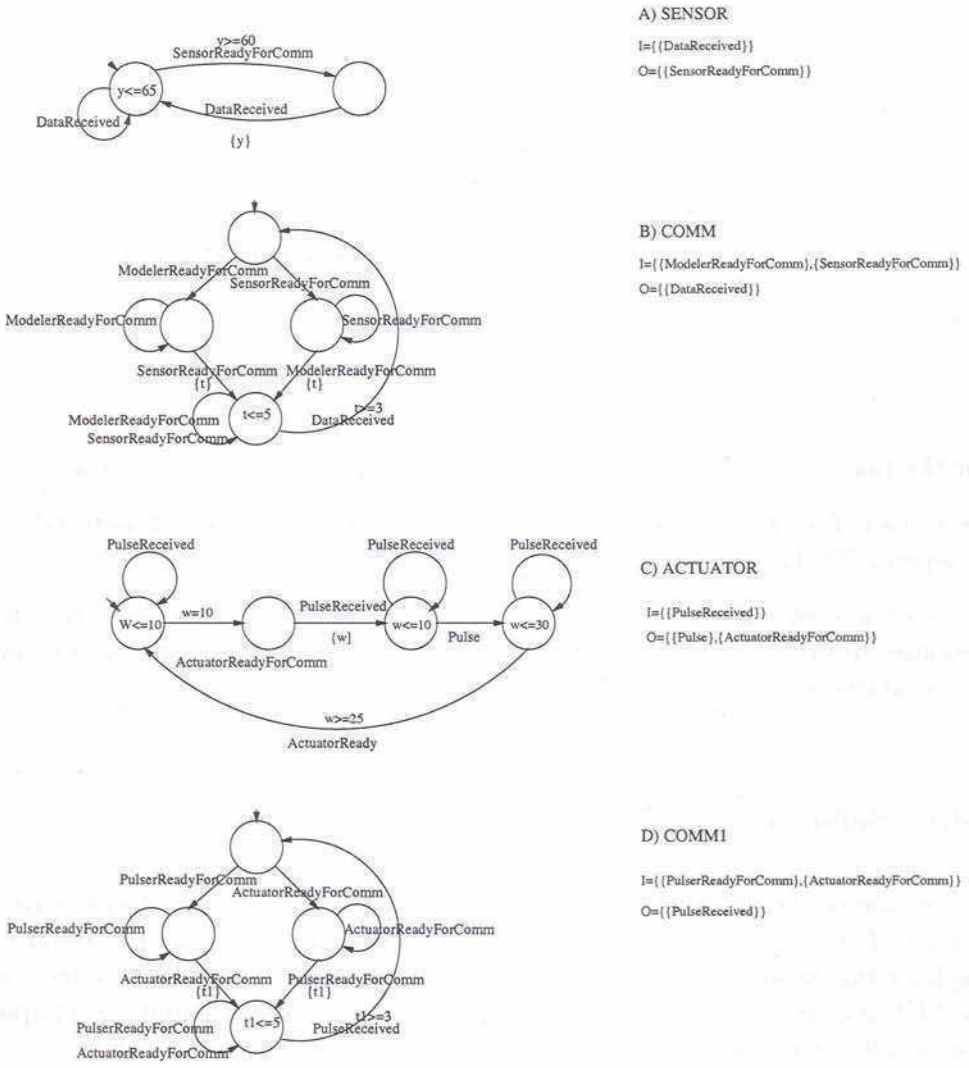
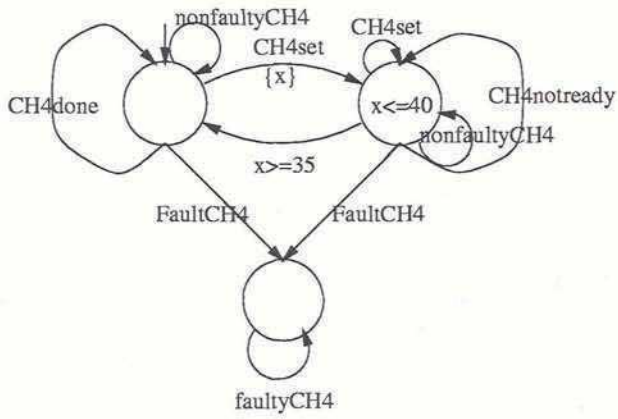


Figure 4.6: The sensor, the actuator, and the communication for the Active Structure System



I = {{CH4done, CH4Notready}, {nonfaultyCH4, faultyCH4}, {CH4set}}

O= {{faultCH4}}

Figure 4.7: Assumptions about CH4 sensor

For the sake of readability, stuttering edges are not shown in the figures.

We present four types of requirements using observers, namely: Safety Observers, Büchi Observers, TCTL observers, and LDI observers.

Experience shows us that safety observers are enough for most requirements (bounded response, freshness, etc.). However, we want to illustrate how other interesting properties can be stated (and verified) over the design.

4.4.1 Safety Event Observers

In the case of the Safety Observers, the underlying timed automaton evolves to an *Error* location if the behavior of the system entails a potential invalidating scenario. Therefore, checking the requirement reduces to checking reachability in the parallel composition of the *SUA* and the observer. In next sections we will see how to apply techniques to do this in a rather efficient way.

According to our experience, automata based notations turned out to be simpler than most logics for describing expected sequences of events. By using TA observers, it is easy to express complex temporal requirements for control/data flows, to detect invalid use of bounded capacity objects as queues, lost of signals, etc. Similar ideas have also been applied in the untimed framework ([52, 84, 1], etc.) and in the timed one ([142, 58], etc). Other operational timed languages based on process algebras ([137, 127, 155, 162, 115], etc.) could also be used but we prefer to use a notation closely related to our formal kernel. Observers provide us a nice notion to develop our results, they are also reasonable mean

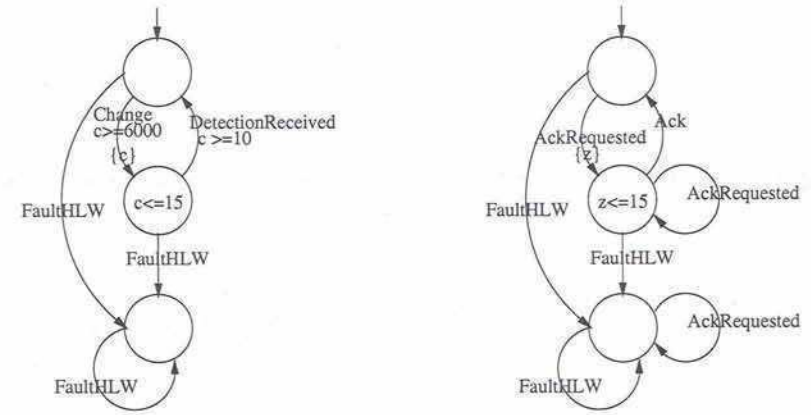


Figure 4.8: Assumptions about HLWLevel sensor

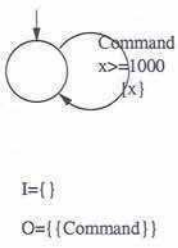


Figure 4.9: Assumptions about the operator

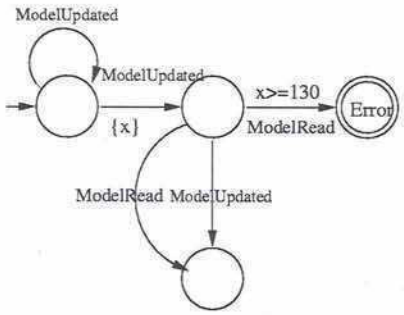


Figure 4.10: Observer for Freshness: From a read to an update of the model

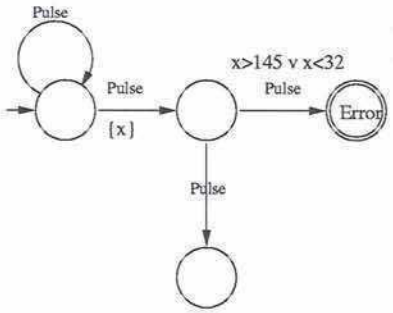


Figure 4.11: Observer for The Regularity: InterPulse Delay

to express requirements in terms of event sequences. [6]

We should remark that some of the properties are expressed in terms of externally non-observable design events. Sometimes, this has to due to the nature of properties: for example, freshness involves read and write operations. Other properties, although they can be informally formulated as a "black box" property needs to be formally formulated on solution-level events. For instance, for some bounded response requirement it is necessary to map it in terms of the actual data/control flow to ensure the correspondence between incoming stimulus and outcoming responses.

For the example of the Active Structural Control System, we show in Fig. 4.10 and Fig. 4.11 the safety observers for the freshness constraint and the regularity property respectively. The Freshness automaton captures cases where the model read by the pulser is older than 130 t.u. The regularity observer captures the scenarios where the pulse are separated by more than 145 t.u. or less than 32 t.u.

To illustrate the flexibility of safety observers, we also map all the presented requirements in terms of data and control flow of our detailed architecture for the Mine Pump example.[7]

[6] However, we are conscious that an user-friendlier notation built on our observers could be provided.

[7] We also use minimum structural information to simplify some observers. For example, we use the fact that between a device *set* and a device *read* of a sensor there must be a *done*.

Separation: Fig. 4.12 shows an observer automaton that accepts all the scenarios where the temporal distance between two consecutive readings of Water-Flow sensor is not in the interval [960 ms ; 1,040 ms].

Bounded Response1: Fig. 4.13 shows an observer automaton that accepts all the scenarios where the proposed solution does not respond to High and Low water level changes setting or clearing the motor within 200 ms (supposing CH4 levels are ok). Note that it also captures as errors cases where two detections arrives before the data is read (therefore, we also check that data is not overwritten).

Bounded Response2: Fig. 4.14 shows an observer automaton that accepts all the scenarios where it is violated the deadline of 200 ms for disabling the pump when CH4 goes high.

Bounded Response3: Actually, this requirement was broken into two subgoals in terms of this design: firstly, any presence of high CH4 levels shall be translated into the "AddAlarm" operation of the console proxy object within 200 ms., and secondly, the operator shall be informed within 350 ms of any operation on the console proxy object. Fig. 4.15 shows an automaton that accepts all the scenarios where the "AddAlarm" operation of the console proxy is not executed within 100 ms of critically high CH4 value though the sensor device is working. Fig. 4.16 shows an observer automaton that accepts all the scenarios where the operator is not informed of an operation performed on the proxy within the 350 ms limit.

Freshness 1: The observer automaton of Fig. 4.17 would detect scenarios where CO readings are more than 100 ms old in normal circumstances (i.e., "notready" event does not occur).

Correlation: Fig. 4.18 shows an observer automaton that accepts all the scenarios where CH4 and CO readings, which are paired into the logger, are values which ages differ more than 100 ms.

False Alarm CO: Fig. 4.19 shows an observer automaton that detects all the scenarios where a "faulty-device" alarm goes off although the CO device is working properly.

Fault Detection HLW: Fig. 4.20 shows an observer automaton that accepts all the scenarios where the high-low Water level device fails and the alarm is not informed to the console proxy within 1650 ms. Note that we are assuming that any operation on the console proxy shall be reflected into the console display within 350 ms. (Fig. 4.16).

False Alarm HLW: Fig. 4.21 shows an observer automaton that detects all the scenarios where a "faulty-device" alarm goes off although the HLW device is working properly.

Fault Detection NET: Fig. 4.22 shows an observer automaton that accepts all the scenarios where a failure in the net is not informed to the remote controller within 2000 ms.

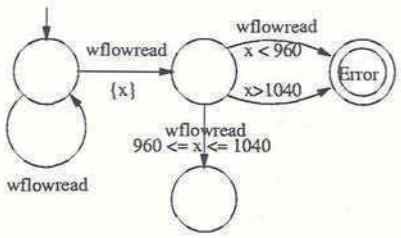


Figure 4.12: Observer for Separation

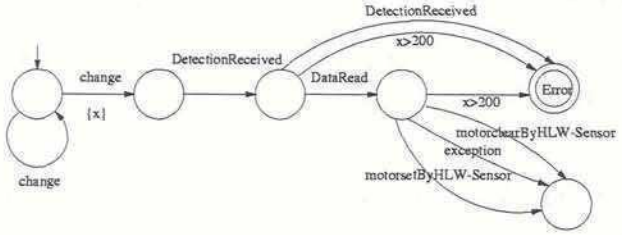


Figure 4.13: Observer for Response1

Freshness2: Fig. 4.23 shows an observer automaton that accepts all the scenarios where the data at CH4 status is older than 100 ms.

Object Interface Assumptions: Fig. 4.24 shows an observer automaton that accepts all the scenarios where more than four alarms are informed by the CH4 Sensor within two Get Package Operations.

In the next sections, we present other approaches to describe and verify properties not expressible using the formerly presented reachability approach.

4.4.2 Büchi Observers

This is a natural extension of the observer technique used in the former sections. This notion is a event-oriented version of Büchi TA presented in [7]. Büchi TA are real time extensions of Büchi automata [44].

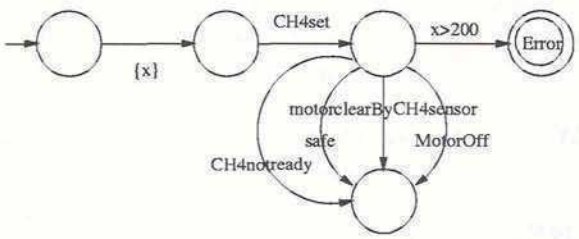


Figure 4.14: Observer for Response2

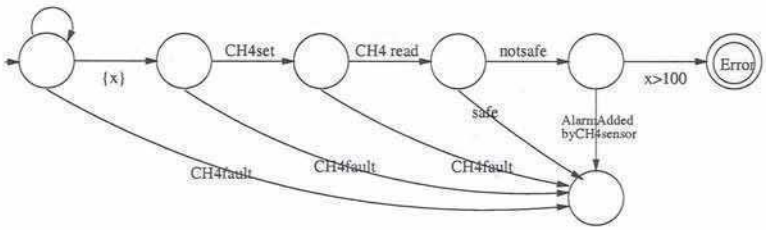


Figure 4.15: Observer for Response3: From CH4 to Console Proxy

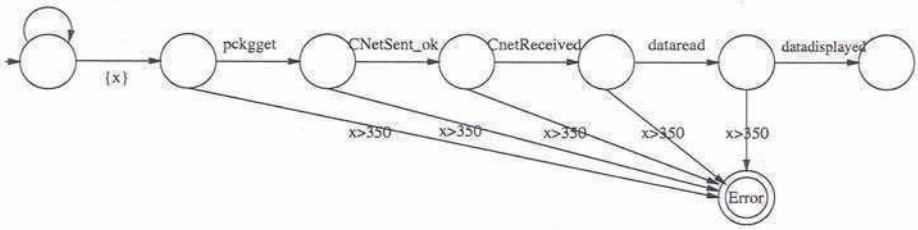


Figure 4.16: Observer for Response3: From Console Proxy to ConsoleDisplay

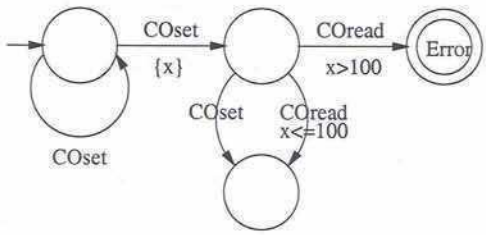


Figure 4.17: Observer for Freshness 1

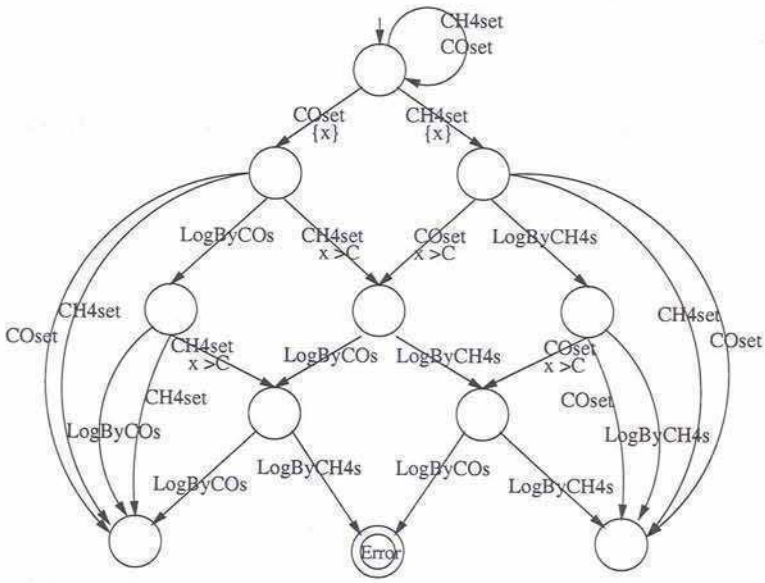


Figure 4.18: Observer for Correlation

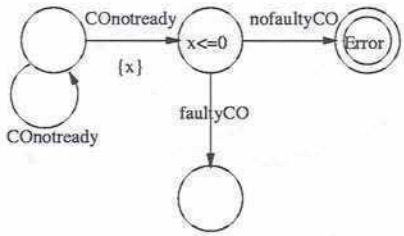


Figure 4.19: Observer for False Alarm for CO

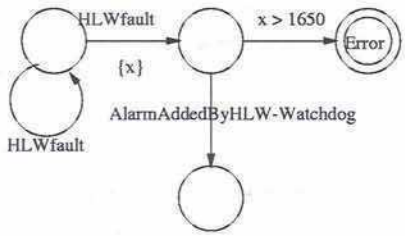


Figure 4.20: Observer for Fault Detection HLW

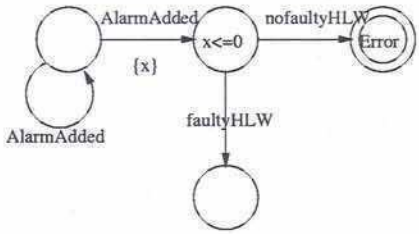


Figure 4.21: Observer for False Alarm for HLW

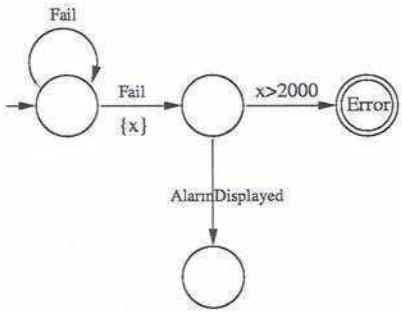


Figure 4.22: Observer for Fault Detection NET

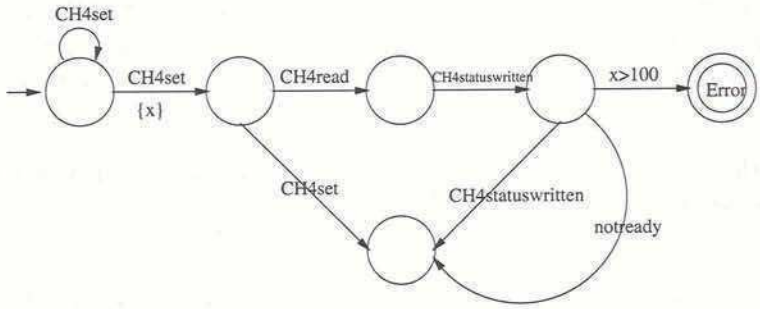


Figure 4.23: Observer for Freshness 2

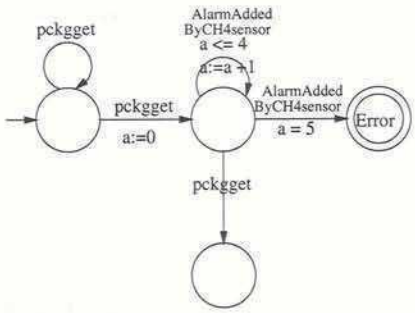


Figure 4.24: Observer for Console

Definition 21 *A Büchi TA B is a pair (O, F) where O is TA and $F \subseteq Locs(O)$ (repeating locations).*

A run σ of O starting from the initial state is *accepting* for B if for every position $k \in \Pi(\sigma)$ there exists $i \in \Pi(\sigma)$: $k \preceq i$ such that $\sigma(i)^{@} \in F$ (i.e. some locations of F are visited infinitely many times). The language of B denoted as $L^{\infty}(B)$ is the set of all timed label-sequences with a supporting accepting non-zeno run for O (i.e., a subset of $L^{\infty}(O)$).

Definition 22 *A Büchi observer B for a component A is a pair (O, F) where O is an observer for A and $F \subseteq Locs(O)$ (repeating locations). Obviously, the language of B, $L^{\infty}(B)$ is the set of all timed label-sequences with a supporting accepting non-zeno run for $[O]$ (i.e., a subset of $L^{\infty}([O])$).*

Given a TA A and a Büchi observer B, the problem we want to solve is to know whether $L^{\infty}(A) \cap L^{\infty}(B) = \emptyset$. As remarked in [149] our notion of satisfaction is based on language intersection rather than the usual automata-theoretic definition based on language inclusion. Since non-deterministic Büchi TA are not closed under complementation, the problem of inclusion is generally undecidable [2]. However, the problem of intersection is decidable. Unlike, [149] our approach is event based (i.e. we do not look at traces but the execution of transitions). Since we aim at language intersection when trying to prove

that the system satisfies a property ϕ we have to use Büchi Observer B expressing the negation of ϕ which for most interesting properties can be found easily in practice [149]. In general, the can be useful for expressing liveness properties ("something good eventually happens").

Note that this generalizes the safety observer framework of previous sections where F is just the *Error* state.

Example: 12 *Figure 4.25 shows a Büchi Observer for the Mine Pump Design that captures the following requirement: There is no more than two Command requests not served in the system and every command is eventually served. Actually, the first part can be expressed (and solved) by using the safety observers. We have a TA for each situation of the command (being the first in the system or arriving while there is another command not served yet).*

4.4.3 TCTL Observers

Interesting properties can be stated by using an observer and a TCTL formula: Given an observer automaton O for a component *SUA*, a labeling function $P : Props \mapsto 2^{Locs(O)}$, and ϕ a TCTL formula on *Props* the designer can ask whether $[SUA \parallel O] \models_{\mathcal{P}^*} \phi$ where $\mathcal{P}^*$ is the natural extension of $\mathcal{P}$ to the cartesian product.

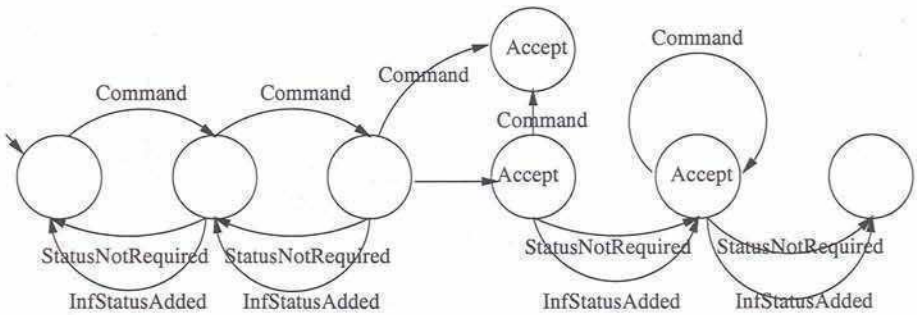
Note that by using observers we avoid requiring the designer to label states of *SUA*, this is important from a methodological point of view since designers should be unaware of the way the system is mapped into the Timed automata model (Chapter 5).

Example: 13 *In the Mine Pump example we want to know whether the system can always be driven to a safe state. The observer shows three different levels of risk states due to water level. The higher the level the more it takes the system to normalize after the corrective action (motor on). The TCTL query is then:* $\forall\Box(Danger \rightarrow \exists \Diamond_{\leq k} Normal)$ *(mode changes). See Figure 4.26.*

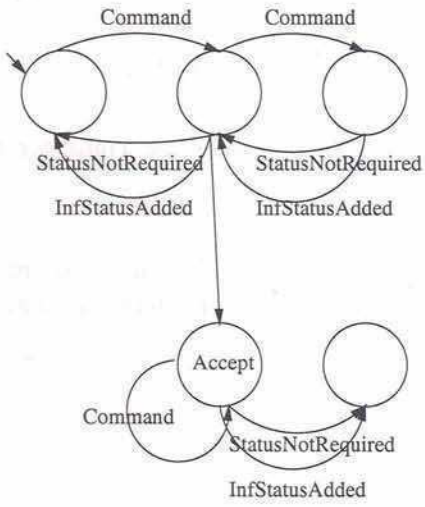
4.4.4 Linear Duration Observers

To check duration properties over the *SUA* we can also use observer components. That is, given an observer automaton O with final locations F, a labeling function $P : Props \mapsto 2^{Locs(O)}$, and ϕ a LDI formula on *Props* we want to check whether $S \parallel O \models_{\mathcal{P}^*} (\phi, F^*)$. Where F^* and $\mathcal{P}^*$ are the natural extensions of F and $\mathcal{P}$ to the cartesian product.

Example: 14 *In Figure 4.27 we see an example for the working example. We want to check that the accumulated time of the motor working though the water level is normal is*



(a) More than two commands stored or command stored but not served.



(b) Command recived but not served

Figure 4.25: Büchi Observer: Responsiveness

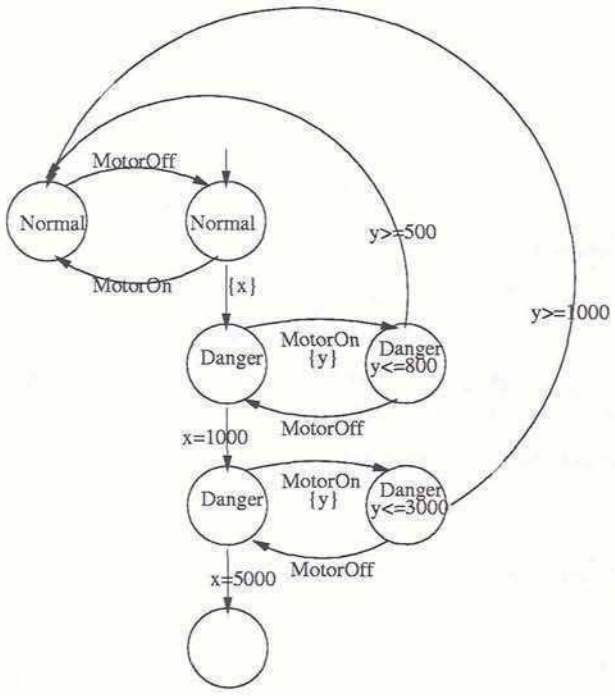


Figure 4.26: TCTL Observer: From Dangerous State to a Normal One

at most 5% of the total amount of time in a window greater than 2000 time units:

$$(20 \int (ON \wedge NORMAL) - \int True \leq 0, \{5\})$$

In the example we are assuming that the HLW-sensor issues the right command when it detects the change of level.

Note that we can also ask whether there is a run which reaches an observer location such that it also satisfies a duration constraint (a satifiability problem) by just negating the comparison.[8]

[8] By using satisfiablity we can ask things like whether it is possible for a system to fail though the fail rate of a faulty component is not higher than a certain value.

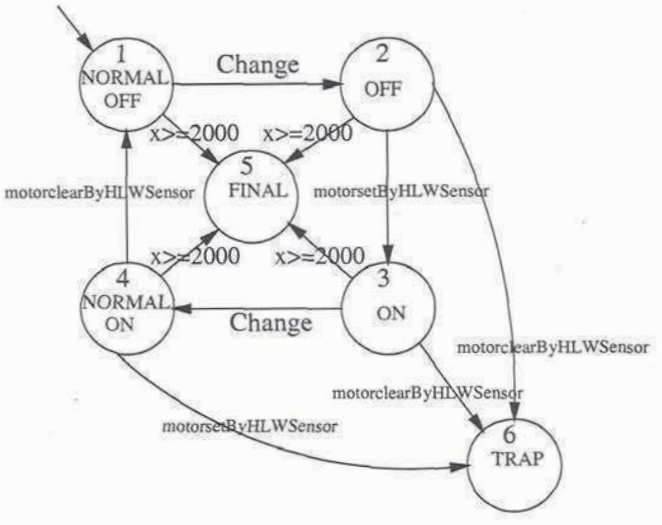


Figure 4.27: LDI Observer: Energy Wasted

Chapter 5

Semantics of Tasks in terms of I/O Timed Components

In the last chapter we show how designs and properties in our proposal can be expressed by means of four elements: (a) task dynamics, (b) environment behavior (c) connector dynamics, and (d) observers. Note that, all but the CDAGs for tasks are essentially I/O Timed Components. The semantics of tasks in terms of I/O Timed Components -also used to perform the model-checking- can be given in three steps. Firstly, we show how the CDAG is embedded into an automaton that reflects the cyclic or sporadic nature of the task. Secondly, TA model is obtained by adding timing information by means of clocks. Finally, those TA can be attached with an I/O interface to become an I/O Timed Components.

5.1 The Untimed Semantics

The CDAG of a task is used as the basis to build the timed automaton that models the complete behavior of the task. Let us explain the final graph in terms of the underlying CDAG. In next section we explain how timing constraints of the referenced figures are derived.

- Periodic Tasks: A location labeled *Idle* replaces the leaves of the CDAG . *Idle* location is connected to the root of the underlying DAG (i.e., the first sequence of actions) (see Pulser task in Fig. 5.1). That edge represents a new release of the task when the period is completed.
- Sporadic Tasks: A location labeled *Waiting for Replenish* replaces the leaves of the CDAG. That location models the wait for the replenish time. This location is then connected to a location labeled *Idle* (see the TA structure of the ACK-Handler in Fig. 5.2). There is an edge from *Idle* to the root of the DAG. This edge is labeled

with the event to be served (triggering event). This event is ignored in any other location by just looping since replenish time has not arrived yet. In Fig. 5.2 we also show how to model the fact that there is a hardware queue for accumulating at most one interruption: we add a variable to be set to one whenever an interruption arrives while the signaling mechanism is disabled. Right after the replenish time has elapsed control jumps directly to the root location of the underlying CDAG in case there is one accumulated interruption; otherwise it jumps to the *Idle* location to wait the next interruption. It is also possible to model that the latching mechanism is enabled only when the server is waiting for replenish time (see Modeler task in Fig. 5.1). If the task has an initialization code the corresponding CDAG precedes the former construction. That is, the final locations are replaced by the location labeled Idle (see Modeler task Fig. 5.1).

5.2 Timed Model

This section describes the way tasks are represented by TA. The key idea of this work is to use TA to analyze a conservative abstraction of system behavior. Following the "separation of concerns" criterion, we exclude the fact that processing resources are shared, but leave its influence on the temporal behavior of the system. The scheduler is not represented in our models. Each task is modeled as running in a dedicated processor but taking into account slowdowns previously calculated using the appropriate scheduling theory (WCCT calculus). It turns out that a conservative and compositional model can be built by just using time constraints as maximum and minimum distances between events. No accumulation is needed to model executed time like in previous dense-time approaches based on Hybrid Automata [15, 38, 152], etc. In this section, we show how to add timing constraints to our untimed models.

Let e be an edge of the underlying CDAG C and let $src(e)$ be its source location. We know that, the label of e stands for one of the final events generated by the sequence of actions associated with $src(e)$ denoted as $Actions(src(e))$. To express the Best-Case occurrence time (BCCT) of this edge, we use a clock, for example x, which is reset at every incoming edges of $src(e)$ and tested at e as $Guard(e) = x \geq D$ where $D = \lfloor \{\Sigma_{i=1}^{\#Actions(src(e))} Actions(src(e))[i].Min\} \rfloor$ where $\lfloor\rfloor$ is floor integer part [1] (the minimum execution time to perform -with no interference- the associated sequence of actions ended in the event of e). To express the deadline for a sequence of actions locations, we use a clock, for instance z, reset at the edge that links *Idle* to the root location (the release instant of the task). We add the following invariant to all location l of the underlying CDAG: $z \leq d$ where $d = \lceil Max\{WCCT(p)/p = Actions(l_1)\&...\&Actions(l_n) \wedge l_1...l_n = l \in Paths(Dag(C))\} \rceil$ (i.e., p is the concatenation of actions of a path) where $\lceil\rceil$ is the ceiling integer part. This

[1] Another option to translate these bounds into integer constants is scaling up all values by multiplying them with a suitable constant. However, it must be taken into account that the complexity of most model-checking algorithms is sensitive to the value of the maximum constant.

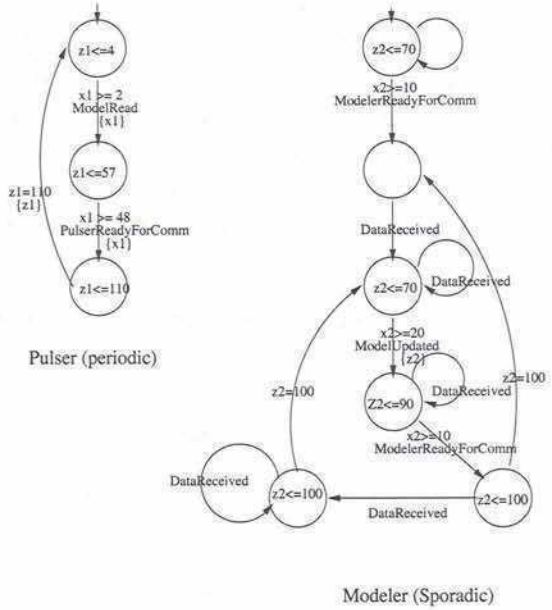


Figure 5.1: Semantics for the Modeler and the Pulser

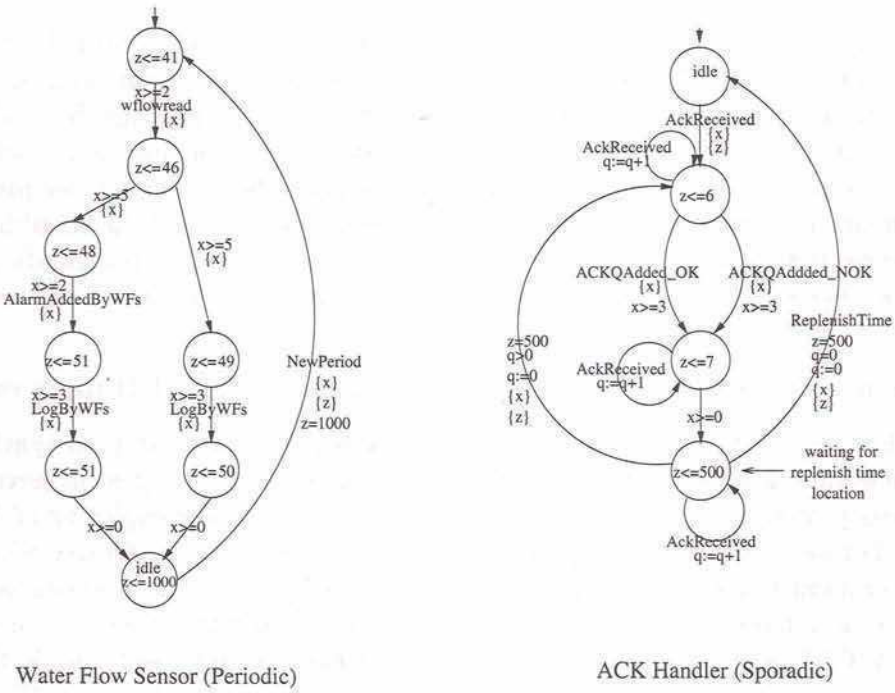


Figure 5.2: Semantics for Tasks

means that the control can not remain at that location more time than the maximum WCCT of the last action associated with that location (WCCT are measured from the release time). That is, the invariants at the control location of the figures are the WCCT for the underlying associated actions (and constitutes the deadline for the outcoming edges). We also use this clock z to control the periodic release by adding an invariant to the *Idle* location $z \leq P$ where P is the period of the task, and associating the guard $z = P$ to the edge that links *Idle* location to the root (see Figures 5.2, 5.1). A similar mechanism is used for controlling the time elapsed at the *Waiting for Replenish* location and jumping to the *Idle* location (see Fig. 5.2,and Fig. 5.1).

We can conclude that the model can be understood as the untimed formal model enriched by the following temporal rules:

- Releases of jobs of a periodic task with period T occurs exactly each T t.u.
- Sporadic tasks serve signals when the corresponding minimal interarrival time has elapsed since the previous served signal.
- An event which stands for the completion of a sequence of actions occurs at most at the WCCT of that sequence of actions relative to the release of the job.
- An event which stands for the completion of a sequence of actions occurs not earlier than the starting instant plus the minimum execution time of that sequence.

Note that this semantic model combines independently the temporal behavior of all tasks. It may be the case that some combination of behaviors are not feasible in the actual implementation due to scheduling secondary effects (e.g., it may not be possible to see all tasks exhibing in a short interval their worst case completion times). However, we believe that conservative model is useful in most situations. Indeed, we may lose properties which are true in the actual implementation due to these side effects of the scheduler; in general, we believe that designer should not relaiant on complex hiden constraints (in our future work section we sketch how we could address some known relations between completion times).

In lemma 6 we show that the resulting model can be see as an I/O timed component.

Therefore, to build the formal model of the system it is necessary to synthesize WCCT for the events associated with the edges of task CDAG as explained in next section. The complexity of the explained procedure is dominated by the complexity of WCCT calculus. Formally, the complexity of the translation procedure is $O(s.n_r.n^2.L)$ where s is the maximum number of locations plus changes of priority (i.e., actions followed by an action of a different priority) of CDAG when converted into a tree (those are the points which WCCT is required by our method or by the calculus itself), n_r is the number of relevant tasks (i.e., the tasks needed to perform the verification; see Chapter 6), n is the maximum number of tasks sharing a same processor than a task to be analyzed (i.e, $Max(\#A_k / \exists \tau_k \text{ to be analyzed} \wedge A_k = \{\tau_i / Processor(\tau_i) = Processor(\tau_k)\})$, L is the

ratio of the longest task period to the shortest task period. In practice, internal task descriptions are quite simple and the complexity is also insensitive to L (see [86]). Therefore, s,l are very small the total number of tasks becomes the scalability parameter of the complexity term (polynomial).

Lets conclude that the semantics of a CDAG is indeed an I/O timed component.

Lemma 6 *A possible I/O interface for tasks-TA is* $I = \{\{triggering\ event\}\}$ *if the task is sporadic or the empty set otherwise, and* $O = \{O(l)/l\ is\ location\ of\ the\ task\}$.

Proof 11 *Clearly* $O, I \subseteq P$ *where* P *is a partition of the set of labels of the CDAG. It is also clear that all events of a candidate output selection are enabled at the same time as required by Output selection definition. To show the non-zenoness regardless input transitions of task-TA let us observe some facts about locations of the abstract code:*

By definition (see Sect. 5.3), the WCCT for last action of location l *(*$I(l)$*) is greater or equal than WCCT of its previous location,* l' *(which is* $I(l')$ *or 0 if* l *is the root of the single-rooted DAG) plus the upper bound for the execution time of the sequence associated with* l *(let us recall that in the calculus* $E_{ij+1}(k) \geq E_{ij}(k) + C_{ij+1}$ *). All edges leaving a non-final location* l *are guarded "*$x \geq D''$ *where* $D = \lfloor Min\{\Sigma_{i=1}^{\#Actions(l)}\ Actions(l)[i].Min\}\rfloor$ *and then* D *is less or equal to upper bound for the execution time of the sequence associated with* l*. From all previous observations we conclude that:* $\forall e, e' \in Edges(Dag(A)) : tgt(e) = src(e') : WCCT_e + Delay(e') \leq WCCT_{e'}$ *(where* $Delay(e') = D$ *if* $Guard(e') = x \geq D$ *or* $Delay(e') = 0$ *otherwise,* $I(src(e)) =$ "$z \leq WCCT_1''$, $I(src(e')) =$ "$z \leq WCCT_2''$*). This means that, no matter how late a location is entered (the latest is* $I(src(e))$ *for some incomming edge* e*) time can elapse (0 or more t.u.) in order to satisfy the delay of a outcomming-transition guard and thus perform a discrete transition. We also know that,* $WaitingforReplenish$ *or* $Idle$ *location are sink locations (i.e., runs eventually arrive to them). Those locations can also be abandoned since WCCT of the whole task is less or equal to the period.*

Hence, in the case of periodic tasks, a run can be extended in such a way that it takes infinitely many $NewPeriod$*-labeled transitions. Thus, time diverges. In the case of sporadic task runs can reach the location that waits for the trigger event (the input selection) but in that location time can progress unboundedly.*

We can also conclude that a run containing an infinite number of non-input transitions must contain an infinite number of $NewPeriod$ *or* $ReplenishTime$ *labels - depending of its cyclic or sporadic nature. Those runs must be time divergent.*

Finally, let us prove the following theorem

Theorem 5 *The parallel composition of the model SUA and an observer is non-zeno.*

Proof 12 *We know that tasks are I/O components by Lemma 6. Two tasks Components are clearly I/O compatible among themselves (at most they share a triggering event). In Section 4.3 we required constraining components to be I/O compatible with the tasks and with other constraining TA they synchronize with. An observer component for a given SUA is compatible with all the SUA components. Thus all components are pairwise compatible then, applying Corollary 4 we know that the composition of the SUA with the observer is an I/O Timed Component and then we conclude the non-zenoness of the model composed with the observer.*

5.3 The WCCT Calculus

WCCT calculus aims at assessing the schedulability of a set of periodic tasks running under Fixed-Priority scheduling ([109, 12],etc.). Roughly speaking, the WCCT (or response time) for a task τ_i can be calculated as the least fixed point of an equation. That equation takes into account the worst-case interference of tasks running on the same processor with higher or equal priority plus the blocking time due to lower priority tasks accessing to mutual exclusion areas.

In our model, each job (invocation) of a task may perform a different path of its CDAG. A path is a sequence of subtasks. A subtask is just a set of actions with a common priority. In fact, the sequence of actions associated with a location can be broken into subsequences of actions which share the same priority. Therefore, each path of the CDAG is a sequence of the subtasks associated with each location of the path. We adapt the technique of [86] to suit our assumptions. As was previously mentioned, that technique helps us to calculate the WCCT of subtasks which may be required for building the timed model presented in Sect. 5.2. In what follows, we recall some notions and results of [86] while pointing out the way we have instantiated or have generalized that work.

Formally, in our framework, a task τ_i consists of $p(i)$ paths. Each path β_h^i of τ_i ($1 \leq h \leq p(i)$), consists of $m(i,h)$ subtasks $\beta_{h,1}^i ... \beta_{h,m(i,h)}^i$. In particular, a subtask $\beta_{h,j}^i$ is characterized by $P_{i,h,j}$ which is the priority of subtask actions, and $C_{i,h,j}$ which stands for its worst case computation requirement (it is just the sum of the maximum requirements for each subtask action). We do not need a deadline since we are only interested in computing the worst case response time (or WCCT) of subtasks.

Definition 23 *A τ_i-idle instant is any time t such that*

- *all work of priority $Pmin_i$ (the minimum priority of any subtask of τ_i) or higher started before t, and*
- *all τ_i jobs also started before t have been completed at or before t.*

Definition 24 *A τ_i-busy period is an interval of time [A,B] such that*

- *both A and B are τ_i-idle instants, and*
- *there is no time $t \in (A,B)$ such that t is a τ_i-idle instant.*

From the fact that in our model $Pmin_i$ (the base priority of τ_i) is the priority of the first subtask of the task (which is indeed the first subtask of all task paths), we easily obtain the following result from Theorem 4 of [86]:

Theorem 6 *The longest response time for any subtask $\beta^i_{h,j}$ is found during τ_i-busy period initiated by τ_i.*

To calculate response times, tasks running on the same processor are placed into groups based upon the priorities of their subtasks relative to τ_i. The five groups of tasks defined in [86] are reduced in our case into three groups (corresponding to types 1,4, and 5 of [86] respectively. Types 2,3 are not possible due to our assumption that $Pmin_i$ is the priority of the first subtask):

1. Higher priority tasks: Task τ_k belongs to this type if all its subtasks have priority equal to or greater than $Pmin_i$ (i.e. $Pmin_k \geq Pmin_i$). They can preempt τ_i more than once per τ_i-busy period. The phasing that produces the largest response time for any job of τ_i is to have such a task initiated at the same instant that τ_i is initiated.
2. Lower priority tasks which have a blocking segment: Task τ_k belongs to this type if $Pmin_k < Pmin_i$ but there is at least one subtask with priority greater or equal than $Pmin_i$. Only one sequence of consecutive subtasks of priority equal to or higher than $Pmin_i$ (an H segment [86]) of one of this type of tasks may influence the response time of τ_i (precisely that segment is running when τ_i is initiated). The length of the longest H-segment is called the blocking term of i (denoted B_i).
3. Lower priority ones: Task τ_k belongs to this type if all its subtasks have less priority than $Pmin_i$ (i.e. $\forall 1 \leq h \leq p(k), 1 \leq j \leq m(k,h) : P_{khj} < Pmin_i$). All subtasks have less priority than $Pmin_i$. Therefore, they can not influence the response time of τ_i.

Taking into account that we need to deal with only three groups of tasks, we adapted the original technique to directly obtain the WCCT for intermediate subtasks without "canonization".[2]

Therefore, these are the steps to follow for analyzing τ_i subtasks:

[Step 1:] Determine the length of the τ_i-busy Period:

$$L_i := min(t > 0 / B_i + \lceil t/T_i \rceil * C_i + \sum_{p \in MP_{i11}} \lceil t/T_p \rceil * C_p = t).$$

[2] Canonization is a technique presented in [86] that changes priority of subtasks without altering the WCCT of the last subtask.

Where:
$Pmin_i$ is the base priority of task τ_i (remember tasks should start and end with the base priority which is the minimum priority of all subtasks).
$MP_{i,h,j}$ is the set of tasks that can preempt $\beta^i_{h,j}$ (i.e., $\{\tau_p/p \neq i \ and \ Pmin_p \geq P_{i,h,j} \ and \ Processor(i) = Processor(p)\}$. Note that, in this formula we refer to $\beta^p_{1,1}$ which is the first subtask of any path of τ_p.
T_p is the Period of task τ_p.
C_p is the computation requirement of task τ_p. In our model, this value is the maximum computation requirement of the path of task τ_p (the computation requirement of a path is just sum of upper bounds of its associated actions).[3]
B_i: is the blocking time due to PCP emulation.

[Step 2:] Analyze the WCCT for each Job.

Let N_i be $\lceil L_i/T_i \rceil$, the number of jobs of the task where τ_i path belongs to be executed during its busy period. Then for all k $\leq N_i$, we must calculate the worst case response time of each subtask in each job to get the maximum in the busy period.
The recursive definition of WCCT is:

$$E_{i,h,1}(k) := min(t > 0/B_i + (k-1) * C_i + C_{i,h,1} + \sum_{p \in MP_{i,h,1}} \lceil t/T_p \rceil * C_p = t).$$

This is the WCCT of the first subtask of task path β^i_h for the k-th job (this is the subtask shared by all paths).

Then, the response time of $\beta^i_{h,j+1}$ for its k-th job is:

$$E_{i,h,j+1}(k) := min(t > 0/E_{i,h,j}(k) + C_{i,h,j+1} + \sum_{p \in MP_{i,h,j+1}} (\lceil t/T_p \rceil - \lceil E_{i,h,n_{i,h}(j,p)}(k)/T_p \rceil) * C_p = t).$$

where
$n_{i,h}(j,p) = max(n \leq j/p \in MP_{i,h,n})$.
Finally, WCCT of $\beta^i_{h,j}$ is $Max\{E_{i,h,j}(k) - T_i * k/1 \leq k \leq N_i\}$. It can be trivially checked, by using the calculated values, that each of the tasks jobs finishes its work before a new period (i.e., the replenish time for sporadic tasks) is started (one of the modeling assumptions). Note that, since paths intersect in common prefixes (at least the root), $\beta^i_{h,j}$ may also appear as $\beta^i_{h',j'}$ for $h \neq h'$ with $j = j'$. However, WCCT calculus provides the same result in both cases, as expected. In fact, the calculus just depends on the sequence of precedent subtasks and those sequences coincide in both paths.

Let us point out that there is a generalization w.r.t. the original work since we do not use canonical form. We calculate $E_{i,h,n_{i,h}(j,p)}$ to know a point up to which all jobs of task τ_p have been executed (to see how many hits were already counted). In fact, we need at most to consider the computation requirements of the rest of the jobs of τ_p (potentially accumulated because of the precedent higher priority subtasks) to calculate the worst case

[3]The paths of the initial code of a sporadic task are considered as part of the set of path of the task

completion time of the subtask $\beta^i_{h,j+1}$. Note that these values $n_{i,h}(j,p)$ can be calculated previously in $O(n_r.s)$ where n_r is the number of tasks to be analyzed, s is the maximum number of subtasks of those tasks.[4] Nevertheless, the previous calculus is more general than required because at most the execution of one job would be pending due to the assumption that worst case response times should be smaller than periods.

[86] shows that the complexity of solving those equations is $O(M.n^2.L)$ [5] where n is the number of tasks M is the number of deadlines to be analyzed and L is the ratio of the longest task period to the shortest task period. By instantiating that complexity calculus and recalling the previous remarks on the computation of $n_{i,h}(j,p)$, it can be straightforwardly concluded that the complexity of this calculus is $O(s.n_r.n^2.L)$ where n_r is the number of tasks to be analyzed, s is the maximum number of subtasks of those tasks, n is the maximum number of tasks sharing a same processor than a task to be analyzed (i.e, $Max_{\tau_k \text{ to be analyzed}}(\#\{\tau_i / Processor(\tau_i) = Processor(\tau_k)\})$), and L is the ratio of the longest task period to the shortest task period.

Associated with a sporadic task, an interruption handler [109] can be also described in our notation. An interruption handler (IH) has a priority greater than any subtask appearing in the corresponding task. At first sight, it seems that this contradicts our assumptions on priority profile (actually the interruption handler followed by the task may become a single preemptive task for other tasks [86]). However, we can treat interruption handler as another task. In fact, we assume that WCCT of the task must be less or equal than minimal interarrival time and thus the IH will be counted at most one time using the calculus presented. Note also that WCCT of the first task of the server ($\beta^i_{1,1}$) is greater to or equal than WCCT of its IH plus $C_{i,1,1}$ since the $B_{IH} \leq B_i$ and $MP_{IH} \subseteq MP_{i,1,1}$ (IH has greater priority than the $\tau_{i,1,1}$, the base priority of τ_i).[6] On the other hand, when analyzing the interference of task τ_i on lower priority tasks, the influence of the interruption handler and the server itself is counted as they were originally treated as a single tasks (both are multiple preemptive with the same period).

Example: 15 *Lets analyze WCCTs of task $\tau_i = ACK - handler$ (Mine Drainage case study) which has a minimal interarrival time of 500 ms. This task τ_i has one path, namely: $([1],12) \rightarrow ([2],13) \rightarrow ([.5],12)$. That is, we want to calculate the WCCT of $\beta^i_{1,1}$,$\beta^i_{1,2}$, $\beta^i_{1,3}$. Note that there exists only one task which is multiple preemptive to this task: the interruption handler of the HLW-sensor which has a computational requirement of 1 ms. On the other hand, the blocking term of τ_i is 2 ms. This is the maximum computational time of the services provided by higher-priority protected objects: "CH_4 Status", "ACK-Queue", and "Log". Therefore the lenght of the τ_i-busy Period:*

$$L_i := min(t > 0/2 + \lceil t/500 \rceil * 3.5 + \lceil t/6000 \rceil * 1 = t) = 6.5$$

the result is 6.5 and thus only one job executes during the busy period (i.e., $N_i = 1$).

[4] each subtask $\beta^i_{h,j}$ may store for each priority $p \geq P_{i,h,j}$ the index $n \leq j$ of the last previous subtask such that $p \geq P_{i,h,n}$. Those values can be stored in one Depth-First traversal of tasks.

[5] when dealing with task that completes before the end of the period like in our case

[6] This property is used to explain why the models we build are non-zeno.

Now,

$$E_{i,1,1}(1) := min(t > 0/ \;\; 2 + 1 + \lceil t/6000 \rceil * 1 = t) = 4$$

$$E_{i,1,2}(1) := min(t > 0/ \;\; 4 + 2 + (\lceil t/6000 \rceil - \lceil 4/6000 \rceil) * 1 = t) = 6$$

$$E_{i,1,3}(1) := min(t > 0/ \;\; 6 + .5 + (\lceil t/6000 \rceil - \lceil 6/6000 \rceil) * 1 = t) = 6.5$$

Since only one job must be analyzed the WCCT of $\beta^i_{1,1}$,$\beta^i_{1,2}$, $\beta^i_{1,3}$ *are* $E_{i,1,1}(1)$, $E_{i,1,2}(1)$, *and* $E_{i,1,3}(1)$ *are respectively.*

Chapter 6

Model Reductions

Complexity of most model-checking methods depends heavily on the size of the model. Normally a design conveys more information than actually needed to prove a certain property. Ad-hoc abstractions, applied by an expert are a common approach to tackle the complexity problem (e.g., see discussion in [142]). To avoid informal claims on the correspondence between the model and the abstraction, some approaches also provide a mathematically-sound framework, but they still require the user to build the abstraction (see for example [55, 83] in the untimed setting). Another group of techniques also provides an automatic mean to check the abstraction given by the user: In [111], a compositional framework is developed where an automata may be replaced with another automata under the assumption of language containment when composed with the same context. The user should provide a candidate abstraction of the automaton and a homomorphism (that preserves timed behavior) as a proof that the abstraction actually holds. That proof is then automatically checked.

Finally, like the authors of [89], we believe that mathematically-sound and efficient abstraction techniques -without requiring too much user ingenuity- are also a paramount issue when developing practical tools. In this group we can found, among others, [89] in the untimed framework or [63, 116] for the timed setting. Following this research line, we provide a set of automatic abstractions for our application model that preserves or enlarges the timed language for our models without compromising non-zenoness.

In the next two subsections, we present an exact abstraction method and a set of rules which produce conservative models. In the first case, a method builds a model which is simulation equivalent w.r.t. the events of the observer (i.e., it is undistinguishable for the observer). In the second case, the rules produce models which can simulate the original one (i.e., it might enlarge the set of behaviors of the original model). Conservativeness means that properties valid in the abstract model are conserved in the original one.

6.1 Exact Abstraction: The Relevance Calculus

Fortunately, in general, it is not necessary to make the parallel composition of all TA that compose the model of the application to observe the relevant system behavior. We exploit architectural information of our models to statically discover a subset of components and events that are sufficient to check whether a requirement is met or not. In [110] it is proposed heuristics to localize a set of process to check a property. Unlike, that work we use our knowledge on the architectural style to build a technique that automatically detects a subset of components to perform a still-accurate analysis (it is not a try and error technique like the one presented in [110]).

Definition 25 *Given a CDAG C and a set of labels R, we define the Relevant Prefix of C for R (denoted $\uparrow^R C$ to be the minimum set of nodes of C satisfying the following set of rules (i.e., Least Fixed Point):*

- $l \in \uparrow^R C$ *if* $O(l) \cap R \neq \emptyset$.
- $l \in \uparrow^R C$ *if* $\exists e : Edges(C) : Src(e) = l \wedge tgt(e) \in \uparrow^R C$.

The irrelevant suffix, denoted $\downarrow^R C$ is the complement set of nodes.

Definition 26 (Relevance Calculus) *Given a set of constraining and task components S, given an observer component O, the set of relevant components and the set of relevant events which are sufficient to perform the verification are the smallest sets RC and RE respectively such that they satisfy the following rules:*

1. *All events of the observer are relevant (i.e., $Labels(O) \subseteq RE$).*
2. *If a task or a constraining component $A \in S$ exports a relevant label which is not an input selection of A of size one then A is relevant (i.e., $A \in RC$).*
3. *All events of a relevant constraining component A are relevant (i.e., $Labels(A) \subseteq RE$).*
4. *The triggering event of a relevant sporadic task is relevant.*
5. *Given a task A and a location l of its underlying CDAG ,then the output selection $O(l)$ is relevant (i.e., $O(l) \subseteq RE$) if there exists a pair of locations $l' \neq l''$ following l (i.e., there exists $e', e'' \in Edges(DAG(A))$ $Src(e') = l$, $Src(e'') = l$, $tgt(e') = l'$, and $tgt(e'') = l''$), and one of them (or both) belongs to the relevant prefix (i.e., $\{l', l''\} \cap \uparrow^{RE} (DAG(A)) \neq \emptyset$).*

Before going into further details, let us sketch the abstraction technique. Firstly, it is calculated the set of relevant components and events with a least fixed-point algorithm

straightforwardly based on the former definition. Secondly, by using reduction rules of Def. 27, the relevant tasks automata are reduced to just show those relevant events as edges. The "submodel" so obtained is guaranteed to have the same behaviors than the whole model, up to the observational power of the observer automaton. That is, the set of (reduced) relevant components generates the same event-sequences captured by the observer than the whole model. Formally speaking, the submodel is simulation equivalent to the original system up to the relevant events. It is important to point out that, differently from former approaches, our method does not necessarily require all tasks belonging to a processing node to check a property that involves just some of those tasks.

Definition 27 (Procedure for Collapsing Tasks Automata) *Given a set of labels R, an I/O timed component modeling a task $C = (A, (I, O))$, then $C_R = (A_r, (I, O_r))$ denote a component where A_r and O_r are obtained from A and O by applying the following rules:*

1. *Eliminate every pair of locations of the underlying CDAG* a *and* b *such that*

   - $\mathrm{b} \neq nil$,
   - *they are in sequence (i.e.,* $\forall e : Edges(Dag([C])) : src(e) = \mathrm{a} \iff tgt(e) = \mathrm{b}$*), and*
   - $\forall e : src(e) = \mathrm{a} : Label(e) \notin R$.

   then, add a location, namely c, *with the same arriving edges than* a *and the same leaving edges and invariant than* b. *Change the delay guards of edges by adding to their constants the minimum delay of edges leaving* a.

2. *Eliminate all nodes l (i.e., locations) of the underlying CDAG which do not reach R-labeled edges in the abstract code (i.e., l in the irrelevant suffix: $l \in \downarrow_R C$).*

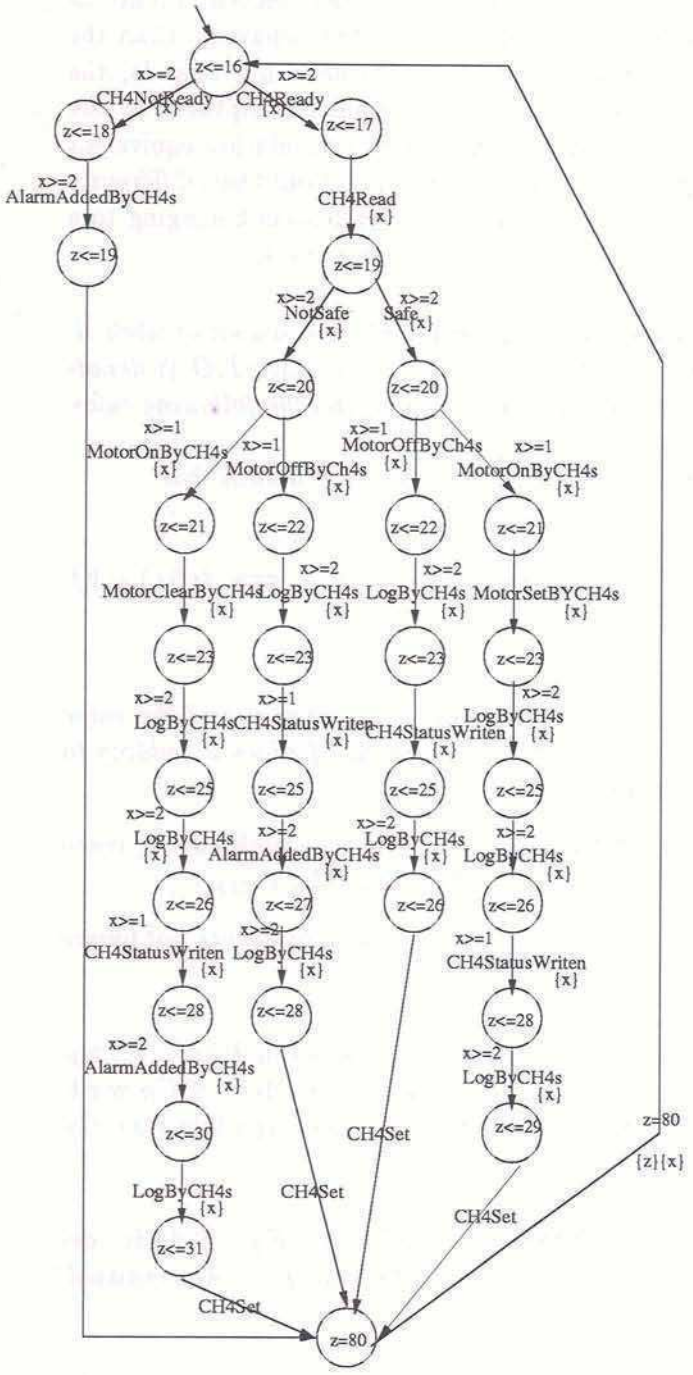
3. *O_r is obtained from O by eliminating the output selections which labels are not longer present in A_r.*

In Sect. 6.3 we show that these rules preserve the semantics up to labels in R (i.e., $[C] \approx_R [C_R]$). Notice that the application of these rules may alter the I/O interface of C_R w.r.t. C by just eliminating output selections. Thus, if C and C' are I/O compatible then C_R and C' are I/O compatible as well.

Example: 16 *In Fig. 6.1 we can see $CH4Sensor$ while in Fig. 6.2 is depicted $CH4Sensor_{RE}$ where RE= {CH4Ready,CH4NotReady, AlarmAddedByCH4Sensor,CH4Read,Safe, NotSafe}.*

We assume that $C_R = C$ in the case that C is a constraining component. That is, in this presentation, we do not provide $\approx_R$ preserving transformation rules for the general case of TA.

The following theorem formally states what is claimed at the beginning of this section:



I={}
O={{CH4NotReady,CH4Ready}, {AlarmAddedByCH4}, {Safe,NotSafe},
{MotorOnByCH4s,MotorOffByCH4torClearByCH4},
{MotorSetByCH4s},{CH4StatusWriten},{CH4Set} }

Figure 6.1: CH4-Sensor Semantics

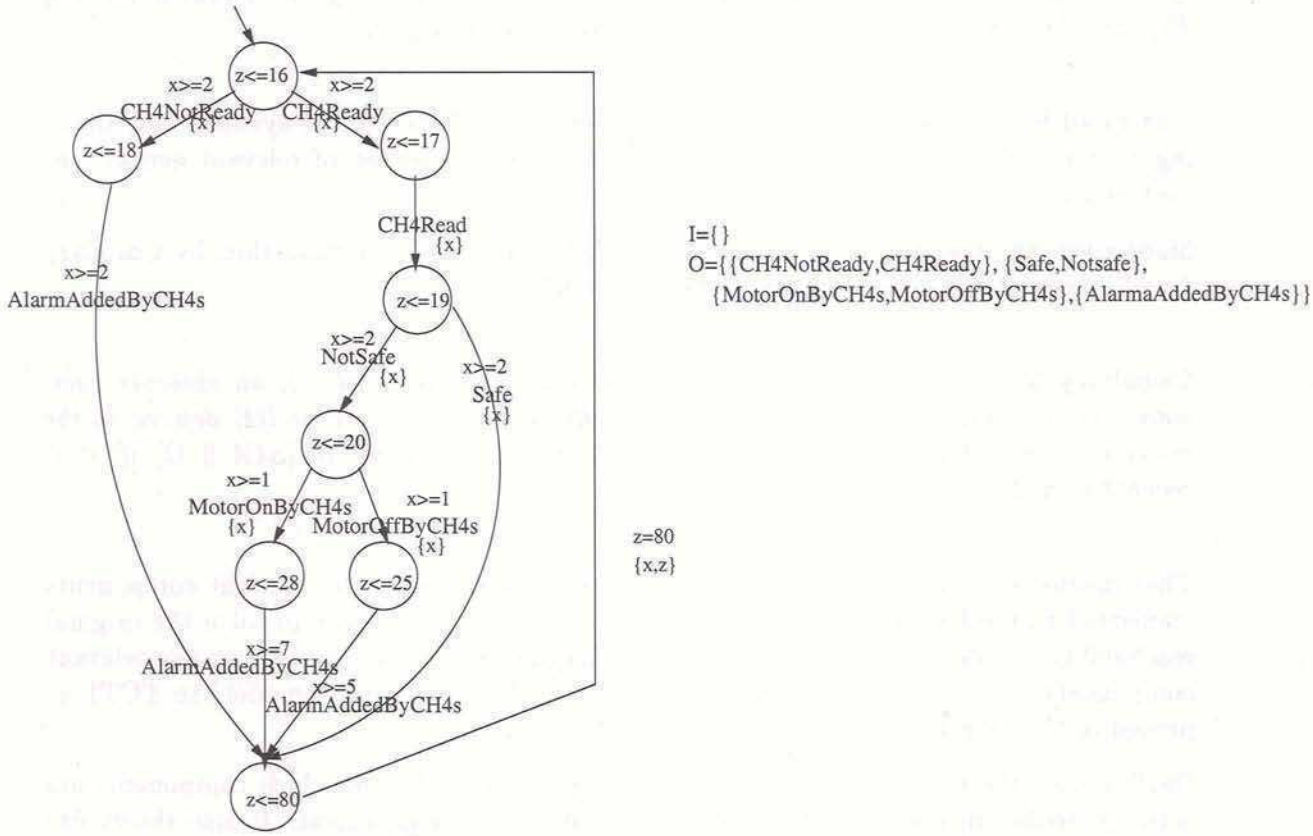


Figure 6.2: CH4-Sensor Semantics collapsed

Requierement	Components	Locs	Edges	Clocks	Abs
Freshness	$Sensor \| \| Comm \| \| Modeler \| \| Pulser$	126	299	5	NO
Regularity	$Pulser \| \| Comm1 \| \| Actuator$	96	186	4	NO

Table 6.1: Components Needed for Requirements of Active Structure System

Theorem 7 *Given a compatible set of components SUA modeling the system under analysis, the set of relevant components RC and events RE defined by the relevance rules of Def. 26, then* $[SUA] \simeq_{RE} [RC]$ *and therefore* $[SUA] \approx_{RE} [RC_{RE}]$.

The proof is done by establishing the CO-bisimulation between the systems and showing that non-relevant components cannot affect the occurrences of relevant events (see Sect. 6.3).

Since given an observer timed automaton O, $Labels(O) \subseteq RE$ by rules then by Corollary 1 and Theorem 1 it can be trivially inferred the following result:

Corollary 5 *Given a compatible set of components SUA (the system), an observer component O, and the set of relevant components RC and relevant events RE defined by the relevance rules of Def. 26 , then the error location is reachable in* $[SUA \parallel O]$ *iff it is reachable in* $[RC_{RE} \parallel O]$.

That means, that we only need to compose the TA obtained of the relevant components translated to the level of abstraction dictated by the relevant events to solve the original reachability problem. Note also that, if the collapsing part is not applied to the relevant components - even branching properties are preserved in the reduced model like TCTL as proved in Chapter 3 and summarized in Chapter 8.

Dealing with the working examples, Table 6.1 and Table 6.2 show which components are actually needed in the parallel composition to solve each requirement. It also shows size parameters of the resulting models (number of locations, edges, and clocks). Experience showed that, these relevant subsets are rather small, reducing the verification effort. Reasons for this encouraging observation are the use of non-blocking communication media and the fact that we do not model schedulers. Therefore, the technique generates rather "loosely coupled and asynchronous" models in terms of the influence among components. In many cases, our relevance calculus discovers that, in order to verify a given observer, it is sufficient to consider the set of components that generates the events named in the observer. Note that even in real size system, the number of interacting components which behavior must be considered for checking a given requirement may be significantly smaller than the number of total components (encouraging the use of this automatic verification approach for real life systems).

Requierement	Components	Locs	Edges	Clocks	Abs
Separation	$WaterFlow - Sensor$	8	10	2	NO
Response 1	$HLWFacts \| HLW - Sensor$	243	1018	4	NO
Response 2	$CH4Facts \| CH4 - Sensor$	108	335	3	NO
Response 3.0	$CH4Facts \| CH4 - Sensor$	83	272	4	YES
Response 3.1	$Console - Proxy \| Net \| ConsoleDisplayer$	267	828	4	NO
Freshness 1	$COFacts \| CO - Sensor$	62	188	4	NO
Correlation	$CH4 - Sensor \| CO - Sensor$	64	138	5	YES
False Alarm CO	$COFacts \| CO - Sensor$	51	149	4	NO
Fault Detection HLW	$HLWFacts \| HLW - Watch. \| AckQ \| Ack - Handl.$	154	521	5	NO
False Alarm HLW	$HLWFacts \| HLW - Watch. \| AckQ \| Ack - Handl.$	294	1034	5	NO
Fault Detection NET	$Con.Proxy \| FailNet \| Con.Displayer \| Timer \| DisplayerWD$	270	1176	6	NO
Freshness 2	$CH4Facts \| CH4 - Sensor$	94	378	4	YES
Console	$CH4 - Sensor \| ConsoleProxy$	84	202	4	YES

Table 6.2: Components Needed for Requirements of Mine Drainage System

6.2 Conservative Abstractions

Most of the research effort was focused on exact abstractions. They are very appealing due to the fact that they are equivalent with the original model up to the property to be checked. However, conservative techniques can be very useful in practice to dramatically reduce time and space in the verification process. To illustrate that other means to reduce the size of the models, we developed a small set of rules to further reduce the size of the obtained TA without eliminating any behavior. This idea is, in some sense, similar to the one of [116] but we are based on the topology of our models and therefore we obtain more reduction. That is, by knowing the topology of the TA modeling tasks, we can state a set of conservative rules to further reduce the size of the underlying DAG, probably loosing accuracy, but without compromising the correctness of the final analysis (they do not eliminate behaviors). Our methodology allows the user to apply these simple rules to further abstract the model. We believe that this is a reasonable way to build an interactive wizard tool that preserves correctness of user manipulation when the full automation becomes infeasible.

To illustrate the concept, we just present two rules (those used in the example):

1. Given three locations a, b, c of the underlying DAG, such that b $\neq nil$, c $\neq nil$, and such that $\forall e : E : src(e) = \text{a} \iff tgt(e) = \text{b} \vee tgt(e) = \text{c}$; then b and c may be eliminated by adding d target of the same set of edges than c and b and source of the union of edges leaving b and c. Its invariant constant must be the maximum between invariant constants of b and c (this rule can be easily generalized to more than two "next" locations). This rule forgets the history of the computation and considers worst case response times of all paths finalizing at the collapsed locations.

2. Given two locations a and b (b $\neq nil$) in sequence perform the manipulation explained in item 1 of Def. 27. Note that, this rule does not add any behavior provided any

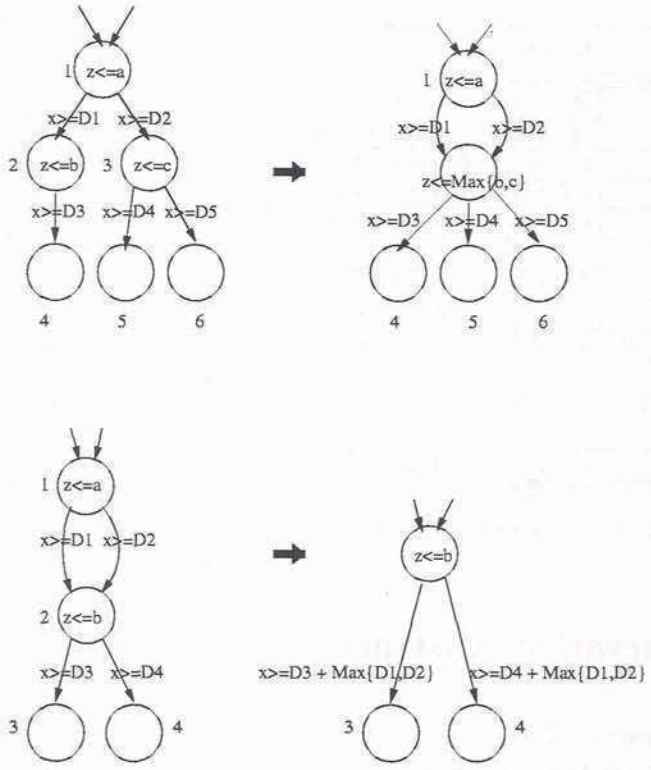


Figure 6.3: Conservative Abstraction Rules.

automaton of the set does not share the labels that are eliminated. Obviously, it does not lose behaviors.

In Fig. 6.3, we show a schema of the transformation rules. In column *Abs* of tables 6.1 and 6.2, we point out in which verifications the former conservative rules were applied. Other conservative transformation rules not used in the example are the elimination of the guards and clocks used to guarantee the minimum delay, or the elimination of any constraining component, etc. It is easy to see that all those rules:

1. Preserve the same language up to the remaining edges. In fact, the obtained Timed Automaton can simulate any run of the original Timed Automaton up to the remaining labels,
2. Preserve non-zenoness. The reason for non-zenoness of these models is that they preserve the following relation: $\forall e, e' \in Edges(A) : tgt(e) = src(e') : I(src(e)) + Delay(e') \leq I(src(e'))$ (see Sect.3.3.1), and
3. Preserve the I/O compatibility - in particular, rule 2 just eliminates output selections or internal edges.

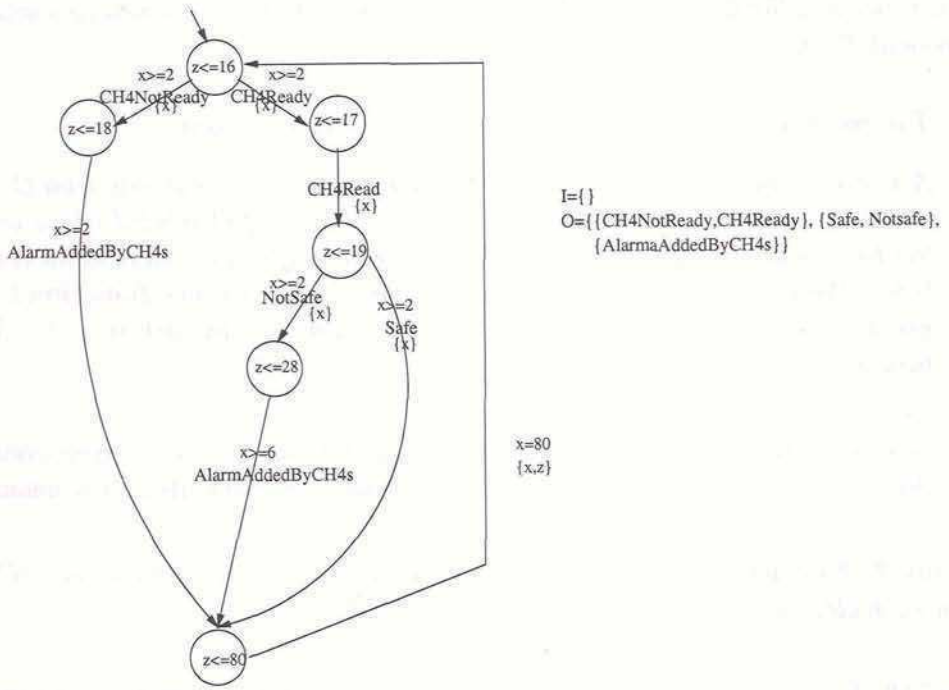


Figure 6.4: CH4-Sensor Conservative Reduction

Example: 17 *In Fig. 6.4 we can see a further abstraction of $CH4Sensor_{RE}$ where two paths where merged into one first using rule one and then rule two.*

6.3 Correctness of Relevance Calculus

Before proving the theorem, let us observe some facts about the relevance calculus.

Proposition 1 (On Including Relevant Components) *The rules include as relevant a component A that synchronizes with a component $B \in RC$ if and only if:*

- *A is a task or constraining component and B is a constraining component such that there is a common label which is not an input selection of A of size one (remember that all events of a relevant constraining component are relevant).*
- *A and B are tasks TA and A triggers B (they can only synchronize trigger events).*
- *B is a task automaton with a location l in the underlying DAG that precedes at least two different locations such that at least one of them belongs to the relevant prefix, and A is a constraining component featuring $O(l)$ as input selection.*
- *A is a constraining component that triggers a relevant task automaton B.*

Then, it is not difficult to see that there are two cases where A synchronizes with a relevant component B and it is not included as relevant by the rules:

1. *The common labels are all input selections of A of size one.*
2. *A is a constraining component synchronizing with an output selection O of B but in all occurrences of O in the CDAG of B the target of O-labeled edges are the same location or the locations belong to the irrelevant suffix (i.e., there is no relevant event before the new release of the task.) Note that the occurrence (and time) of an event just depends on previous events in the same release of the task since the WCCT of a task is less than its period.*

Note also that since all the events of constraining components and trigger events are relevant, the synchronizing labels of $[RC]$ (TA of relevant components) must belong to RE.

Lemma 7 *Automata produced by applying collapsing rules (Definition 27), $[C_R]$ is simulation equivalent w.r.t. the original one $[C]$ up to R.*

Proof 13 *We will prove that the result of applying each rule once produces an automaton which is simulation equivalent to the input automaton. Since $\approx_R$ is an equivalent relation we straightforwardly conclude that the repeated application of rules produces a simulation equivalent automaton. Rule 1:*

Let us define a simulation between $[C]$ and $[C_R]$ when $[C_R]$ differs from $[C]$ in the fact that a, b *were collapsed into* c *(first rule). We say that (s, x, z) S (s', x', z') iff ($s = s'$, $s \neq$ a, $s \neq$ b, $x = x'$, and $z = z'$) or ($s =$ a, $s' =$ c, $x = x'$, and $z = z'$) or ($s =$ b, $s =$ c, $x' \geq x + d$, and $z = z'$) where d is the minimum delay of edges leaving* a.

Now, the only non trivial cases is when s is either a *or* b.

If $(\mathrm{a}, x, z) \mapsto_0^l (\mathrm{b}, 0, z)$ then $(\mathrm{c}, x', z') \mapsto_0^\lambda (\mathrm{c}, x', z')$ and for sure $x' \geq 0 + d$ and $z = z'$ (since $x = x'$ and to jump x must be $\geq d$). Then, $(\mathrm{c}, x, z) \xrightarrow{R}_0 (\mathrm{c}, x', z')$ and obviously $(\mathrm{b}, 0, z)$ S (c, x', z').

If $(\mathrm{a}, x, z) \mapsto_t^\lambda (\mathrm{a}, x + t, z + t)$ then $(\mathrm{c}, x, z) \mapsto_t^\lambda (\mathrm{c}, x' + t, z' + t)$ (the invariant at c *is weaker than the invariant at* a*). Then, $(\mathrm{c}, x, z) \xrightarrow{R}_t (\mathrm{c}, x' + t, z' + t)$ and obviously $(\mathrm{a}, x + t, z + t)$ S $(\mathrm{c}, x' + t, z' + t)$.*

If $(\mathrm{b}, x, z) \mapsto_t^\lambda (\mathrm{a}, x + t, z + t)$ then $(\mathrm{c}, x, z) \mapsto_t^\lambda (\mathrm{c}, x' + t, z' + t)$ (the invariant at c *is the same than the invariant at* b*); then $(\mathrm{c}, x, z) \xrightarrow{R}_t (\mathrm{c}, x' + t, z' + t)$ and obviously $(\mathrm{a}, x + t, z + t)$ S $(\mathrm{c}, x' + t, z' + t)$.*

If $(\mathrm{b}, x, z) \mapsto_0^l (\mathrm{n}, 0, z)$ using an edge guarded $x \geq D$ then $(\mathrm{c}, x', z') \mapsto_0^l (\mathrm{n}, 0, z')$ since $x' \geq D + d$ and $z = z'$ Then $(\mathrm{c}, x, z) \xrightarrow{R}_0 (\mathrm{c}, x, z) \mapsto_0^l (\mathrm{n}, 0, z') \xrightarrow{R}_0 (\mathrm{n}, 0, z')$ and obviously $(\mathrm{n}, 0, z)$ S $(\mathrm{n}, 0, z')$.

It is easy to see that the initial states are related.

Now, let us define a simulation between $[C_R]$ *and* $[C]$ *we say that* $(s,x,z)\ S\ (s',x',z')$ *iff* $(s = s' \neq \mathrm{c},\ x = x',\ z = z')$ *or* $(s = \mathrm{c},\ s' = \mathrm{a},\ x = x' \leq d,$ *and* $z = z')$ *or* $(s = \mathrm{c},\ s' = \mathrm{b},\ x \geq d,\ x' = x - d,$ *and* $z = z')$*, where* d *is the minimum delay of edges leaving* a*. In fact, let* $(\mathrm{c},x,z)\ S\ (s',x',z')$*. If* $(\mathrm{c},x,z) \mapsto_0^l (\mathrm{n},0,z')$ *then* $s' = \mathrm{b}$*,* $z = z'$ *and* $x' = x - d$ *since* $x \geq D + d$*. Also, since* $x' \geq D$ *then* $(\mathrm{b},x',z') \mapsto_0^l (\mathrm{n},0,z')$*.*

If $(\mathrm{c},x,z) \mapsto_t^\lambda (\mathrm{c},x+t,z+t)$ *and* $x < d$*. Then, we know that* $s' = \mathrm{a}$ *and* $x = x'$ *then if* $x' + t \leq d\ (\mathrm{a},x',z') \mapsto_t^\lambda (\mathrm{a},x'+t,z'+t)$ *(remeber that the task TA are non-zeno then* $I(\mathrm{a})[z'+t]$*) and the simulation holds. If* $x' + t > d$ *then* $(\mathrm{a},x',z') \mapsto_{d-x'}^\lambda (\mathrm{a},d,z'+(d-x')) \mapsto_0^e (\mathrm{b},0,z'+(d-x')) \mapsto_{t-(d-x')}^\lambda (\mathrm{b},t-(d-x'),z'+(d-x)+t-(d-x'))$ *then* $(\mathrm{a},x',z') \xrightarrow{R}_t (\mathrm{b},t-(d-x'),z'+t)$*,* $x+t = x'+t \geq d$*,* $t-(d-x') = x+t-d$*, and* $z+t = z'+t$*.*

It remains to be proved: $(\mathrm{c},x,z) \mapsto_t^\lambda (\mathrm{c},x+t,z+t)$ *and* $x \geq d$*. Then, we know that* $s' = \mathrm{b}$*,* $z = z'$*, and* $x' = x - d$*. We have* $(\mathrm{b},x-d,z') \mapsto_t^\lambda (\mathrm{b},x-d+t,z'+t)$ *since the invariants are the same; then* $(\mathrm{c},x+t,z+t)\ S\ (\mathrm{b},x-d+t,z+t)$

Rule 2:

For the rule 2 the simulation is $(q,x,z)\ S\ (q',x',z')$ *iff* $(q = q',\ q \in \uparrow^R Dag(C)$*(the relevant prefix), and* $x = x'$ *and* $z = z')$ *or* $(q' = Idle\ , q \in \downarrow^R Dag(C),$ *and* $z = z')$*. It is easy to see that* S *is actually a bisimulation due to the fact that a) the time of occurrence of the next relevant event in the irrelevant suffix (i.e., NewPeriod , ReplenishTime occurrence) depends on* z *clock value (which is the same in related states), and b)* $[C]$ *is non-zeno and then time can elapse to the next period where all clocks are reset.*

Theorem 7: Given a compatible set of components SUA (modeling the whole system), the set of relevant components RC and events RE defined by the relevance rules of Def. 26, then

$$[SUA] \simeq_{RE} [RC]$$

and therefore $[SUA] \approx_{RE} [RC_{RE}]$

Proof 14 *Let's understand why there exits a CO bisimulation between the systems. Now, we will define a CO bisimulation for a set of labels* R *equal to* RE *plus the new period and replenish time events for all relevant tasks (clearly if we prove it for a bigger set we prove it for the original* RE*).*

The symmetrical bisimulation proposed B *in the union of the sate space of* $[RC]$ *and* $[SUA]$ *is :* $(p,q) \in B$ *iff* $\forall A \in RC$

- *A is a constraining TA then* $\Pi_A(p) = \Pi_A(q)$*.*
- *A is a task TA then either*

- $\Pi_A(p) = \Pi_A(q)$ *or,*
- $\Pi_A(p)^{@} \in \downarrow^R (Dag(A))$, $\Pi_A(q)^{@} \in \downarrow^R (Dag(A))$, *and* $\Pi_A(p)(z) = \Pi_A(q)(z)$, *i.e., in both states the task is at a location of the irrelevant suffix (i.e., locations which next event in R is either the new period or the replenish time events) and with the same value for z clock.*

Note that the initial states are B-related (i.e., $(init_{[RC]}, init_{[SUA]}) \in B$ *and* $(init_{[SUA]}, init_{[RC]}) \in B$*).*

Our reasoning is focused on the relevant components. This can be done due to the following facts derived from the rules:

- *Non-relevant components can not block the occurrence of any relevant discrete-transition (see Proposition 1).*
- *If e is an edge of the CDAG such that* $Label(e) \in O$ *where O is a non-relevant output selection, and e' another edge such that* $src(e') = src(e)$ *and* $Label(e) \in O$*, then either* $tgt(e') = tgt(e)$*, or both target locations belong to the irrelevant suffix.*
- *Any output discrete-transition occurring in any non-relevant component does not synchronize with any relevant component.*
- *No input discrete-transition is mandatory to make time diverge in a component.*

We use the former facts to show that bisimulations are possible under the context of the non-relevant components:

Let $(p, q) \in B$ *with* $p \in G_{[RA]}$ *and* $q \in G_{[SUA]}$.

Since constraining components are in the same local state for global B-related states we concentrate our reasoning on the relevant tasks. If $p \mapsto_0^a p'$ *and* $a \in R$ *then it is easy to see that each of the relevant timed automaton in q can execute the same discrete transition -perhaps executing some irrelevant transitions remaining at the equivalence class- and then it arrives into a B-related state. In fact, the relevant TA are in the same state in p and in q except for the task TA which are at different locations of irrelevant suffixes. In the last case, if the discrete transition involves any of those TA, the transition must be a "new period" or "replenish time" event. In those cases, since the z clock has the same value for both TA they can perform that transition instantaneously after, perhaps, performing some irrelevant transitions and end in a B-related state (the root location, with clocks reset).*

Secondly, if $p \mapsto_t^a p'$ *and* $a \notin R$ *then it is easy to see that each relevant timed automaton in q can execute a run of time length t such that (a) no relevant event appears, and (b) arrives to a locally related state of p' no matter the state and evolution of the non-relevant components. In fact, the relevant components are exactly in the same local state in global state p and in q except, perhaps, for the task TA being in different locations of the irrelevant suffix (with the same value of z clock however). If* $t = 0$ *and the task involved in the*

transition labeled with a non-relevant label a is in its relevant prefix, then the irrelevant discrete transition is necessarily part of an output selection where all edges lead to the same location. Therefore, in q that task automaton is exactly in the same state and any of those output transitions can be taken to arrive to a local state B-related to p' (actually, the same local state). If the task involved is in the irrelevant suffix then that task -which is also necessarily at a location of the irrelevant suffix in q- has the chance to stutter or traverse any edge if possible. In both cases, it will fall into a B-related state. If $t > 0$ and the same tasks are at the same relevant location in p and q respectively, the simulation is clearly possible letting time elapse in q. If the task control is in its irrelevant suffix in q and in p then, due to the fact that z clock is the same in q, the simulation is possible. In fact, this is done by elapsing time from q and changing location into the irrelevant suffix as many times as necessary - remember that both systems are non-zeno.

On the other hand, it is easy to see that in the inverse case ($p \in G_{[SUA]}$ and $q \in G_{[RA]}$) B also behaves as a CO bisimulation (by using similar arguments plus the fact that $[RC]$ is less or equally constrained than $[SUA]$ for RE).

To conclude that, in effect, $[SUA] \approx_{RE} [RC_{RE}]$ note that in the previous lemma we see that $[C] \approx_{RE} [C_{RE}]$ and therefore $[RC] \approx_{RE} [RC_{RE}]$ due to Lemma 2 and using the fact that whenever two TA of $[RC]$ synchronize the synchronization label belongs to RE by the rules (all events of relevant constraining components are relevant indeed). We also know that if two TA are C0-bisimular then they are similar, in this case $[SUA] \approx_{RE} [RC]$. Finally, by transitivity of $\approx$ we obtain the wanted result.

Chapter 7

Reducing the Composition of I/O Components

In this chapter, we develop an orthogonal technique to further reduce the size of the model to be fed into the model-checker. In the previous chapters we see how properties are checked by means of observer components. Those virtual components are composed in parallel with system under analysis (*SUA*) and evolve to different locations according to event occurrences. For instance, in our framework, observers allow designers to check safety linear properties (Sect. 4.4.1) and other kind of properties as we present in Sect. 4.4).

In few words, we develop a technique to calculate the components that may be forgotten at each observer location since their future behavior do not influence the future evolution of the *SUA* up to the observer. In many cases, those remaining sets (the relevant components) are proper subsets of the set of all components. Thus, we build compositions (we call them "quotient") which in many cases are smaller than the standard parallel composition (Sect. 2.4). It is important to remark that this general reduction technique is independent from our modeling approach for RTS Designs.[1] Indeed, quite often, models are analyzed by means of observers.

This section shows (a) the postulates that define a good relevance function (a function that determines the set of components potentially needed in each observer location) and how to calculate a relevance function, (b) how to build the "quotient automaton" based on a relevance function to replace the standard parallel composition (the quotient automaton can also be built on-the fly, that during the verification process), (c) how the parallel composition of the whole *SUA* and the observer is related with the quotient automaton

[1] Relevance was inspired on the rather ad-hoc notion of relevant components presented in the last section. We inherit that name from the last chapter but we extended it to a more general setting. However, the method presented here and the Relevant Components Calculus are not comparable. The method presented at this section has granularity at observer location level. On the other hand, the previous calculus works does not make any distinction at the location level but it uses information about task component topology (the irrelevant prefix). As we will see later we can apply both methods.

(bisimilarity), (d) some derived preserving corollaries, and finally (e) some examples that show the practical impact of these ideas.

As related work we could mention the clock reduction technique of [63] where clock activity and equality are detected by fixed point processes. It is worth mentioning [145] a timed automaton-based method for accurate computation of the delays of combinational circuits. Based on the topological structure of the circuit, a partitioning of the network and a corresponding conjunctively decomposed OBDD representation of the state space is derived. The delay computation algorithm operates on this decomposed representation and, on a class of circuits, obtains performance orders of magnitude better than a non-specialized traversal algorithm.

7.1 Relevance

The core of our technique is a notion of potential 'direct influence" of an automata bevavior over another automata behavior when they are at some set of locations. We could simply say that an automata A influences another automaton B iff ther exist an edge of A and an edge of B sharinng the same label such that their source location are in the considered set of loactions. However, that would lead to a rather big symmetrical overestimation. Thanks to the I/O interface attached to TA (the I/O timed components), we are able to define a set of necessary conditions to assure that a component A has no influence on (is irrelevant for) the behavior of another component B when they are at certain locations (for example, if A performs outputs which are stuttered at B locations). That is, we have a better overestimation of potential influence.

Definition 28 *Given an I/O component A and a TA B, we say that A is irrelevant to B for $S_a \subseteq Locs(A)$,$S_b \subseteq Locs(B)$, denoted $A \overset{\times}{\rightarrow} B(S_a, S_b)$ iff*

- *For all $O \in O_A$ such that $O \cap Labels(B) \neq \emptyset$ then, either $\nexists e$ $src(e) \in S_a : label(e) \in O$ or for all e, $src(e) \in S_b, Label(e) \in O$ then $tgt(e) = src(e) \wedge Reset(e) = \emptyset$, and*
- *For all $I \in I_A$ such that $I \cap Labels(B) \neq \emptyset$ then $\forall e \in Edges(B) : (src(e) \in S_b \wedge Label(e) \in I) \Rightarrow (\forall i \in I : \exists e' : Edges(B) : src(e') = src(e) \wedge Guard(e) = Guard(e') \wedge Label(e') = i \wedge tgt(e) = tgt(e') \wedge Reset(e) = Reset(e'))$ (i.e. no matter which input selection is enabled the same change of state can be performed),*

Note that $A \overset{\times}{\rightarrow} B(S_a, S_b)$ and $S'_a \subseteq S_a$ and $S'_b \subseteq S_b$ then $A \overset{\times}{\rightarrow} B(S'_a, S'_b)$.

Definition 29 *Given a set $A_0, A_1, ..., A_n$ of TA, $l \in A_0$, $1 \leq i \leq n$ $S_l(i) = \{\Pi_i(s) \in Loc(A_i)/s$ is reachable in $A_0 \parallel A_1 \parallel ... \parallel A_n \cap \Pi_0(s) = l\}$. We define $S_l(0) = \{l\}$.*

We can overestimate this set at least by two procedures: (a) $\{\Pi_i(s) \in Loc(A_i)/s \text{ reachable in } A_0 \times A_1 \times ... \times A_n \times \cap\Pi_0(s) = l\}$ (reachable in the synchronized product from the initial locations), and

(b)$\{\Pi_1(s) \in Loc(A_i)/\ s \text{ is reachable in } A_0 \parallel A_i \cap \Pi_0(s) = l\}$. Also, the user can estimate this set. A tool can check whether this assumption is violated by arriving to a location not included in the estimation.

Our goal is to define a relation defining the "Relevant Components" for each of the observer locations. Intuitively, *Rel* is a sort of transitive closure of the direct relevance relation. That function called *Rel* must satisfy the following postulates.

Definition 30 (Postulates about Rel) *Let $\mathcal{I} = [1...n]$ a finite set of index, $\{A_i/i \in \mathcal{I}\}$ a compatible set of components and A_0 a TA ; Let $Rel : Locs(A_0) \mapsto 2^{\mathcal{I}}$. We say that Rel is a correct Relevance function for A_0 iff for all $l \in Locs(A_0)$, $i, k \in I$:*

1. $0 \in Rel(l)$.
2. $Rel(tgt(e)) \subseteq Rel(l)$ *for all* $e \in Edges(A_0) : src(e) = l$.
3. $k \neq i \in \mathcal{I}$, $k \in Rel(l)$ *and* $\neg(A_i \xrightarrow{\times} A_k(S_l(i), S_l(k)))$ *then* $i \in Rel(l)$.

The first postulate includes the components that may have a direct impact on the A_0 behaviors. The second extends the relevance of a component to all locations from which a location where the relevance was detected can be reached (this gives the relevance a funnel-like shape). The third is a transitive closure of the influence relation, naturally the influence on the behavior of the A_0 may be indirect. Note that A_0 could be obtained for example composing the original observer with a subset of components.

Rel can be calculated as the minimum set satisfying those properties (by using a simple least fixed point procedure):.

CalculateRel($A_0, A_1, ..., A_n$) returns *Rel*

// Step 1)I nitilaize Rel to satisfy item 1
For each $l \in Locs(A_0)$
$Rel(l) := \{0\}$
End For

// Step 2.1 y 2.2. Update Rel to satisfy item 2 y 3
// Repeat till a fixed point is reached Rel
Repeat
// Step 2.1
$RelOld := Rel$
For Each $t \in Edges(A_0)$
$Rel(src(e)) := Rel(src(e)) \cup Rel(tgt(e))$
End For
// Step 2.2
For each $l \in Locs(A_0)$
For Each $A_i/i : o..n$
For each $k \in Rel(l)$
// Check influence
If$\neg(A_i \xrightarrow{\times} A_k(S'_l(i), S'_l(k)))$
$Rel(l) := Rel(l) \cup \{i\}$
End For
End For
End For
Until $RelOld = Rel$

End CalculateRel

Let us assume that the algorithm is fed with the original parallel composition. Although, this is not really necessary it provides a good estimation of $S_l(i)$ sets, and allows us to give simple characterization of the extra work that must be done to calculate the relevance function. Let V be the number of reachable nodes of this standard composition [2]

Lemma 8 *The complexity of the procedure to calculate Rel -when it is fed with the standard composition- is polynomial on the number of components n, the number of nodes of the*

[2] Examples has showed us that, in general, full analysis of timed systems becomes intractable even with relatively small Vs (just few thousand of locations), which is not the case for untimed systems. Therefore, it is not unreasonable to build "a priori" the reachable Cartesian product to overestimate $S_l(i)$. However, the parallel composition of A_i with the observer seems to provide good estimations as well and might be a good solution when we do not want to build the whole composition. This might be the case of trying to find a counterexample in a large location space using on the fly techniques.

standard composition V, locations of A_0 ($m = \#Locs(A_0)$), and the maximum number of transitions of a component T.

Proof 15 *In fact the overestimation of $S_l(i)$ for all $l \in Locs(A_0)$ and $1 \leq i \leq n$ can be calculated traversing one time the parallel composition in $n.V$ steps (assuming that we use direct access memory to store for each location l' of each component i and each location l of A_0 if $l' \in S'_l(i)$). In each step of the calculus of the fixed point, the new Rel includes the set of the previously calculated Rel and it incorporates at least one component more in one location. Therefore, it can be iterated at most $m.n$ times. Let us assume that we use dynamic programming to keep the whether or not $\neg(A_i \xrightarrow{\times} A_k(S_l(i), S_l(j)))$. Then for the analysis we can assume that this takes no time and then calculate the complexity of getting those values. Thus, at each step k, for each location l the algorithm checks whether the components not detected as relevant influence the already detected ones. This takes at most n^2 steps (following the previous observation about dynamic programming). Step 2.1. takes at most $T.n$ steps. The comparison between Rel and RelOld can be done in m steps (Rel are produced in a monotonic fashion and thus identity can checked by comparing cardinality of each location). Therefore, the algorithm takes at most $m.n.(m.n^2 + T.n + m)$ steps assuming no cost for the influence check. On the other hand, there are at most $m.n^2$ influence relations that must be really checked. To do such a check, let us suppose that a component provides a label-ordered list of transitions in $O(1)$. Then, given the set of locations $S'_l(i)$ and $S'_l(j)$ the ordered lists of labels that are used in definition of influence can be built (filtering the list of all transitions) and intersected (by merging) in $O(T)$.*

Finally, we can conclude that the complexity of the procedure is $O(n.V + m.n^2.T + m^2.n^3)$.

7.2 The Quotient Automaton

We define how to build the parallel composition according to a *Rel* over A_0.

First we define a function that takes a location of the parallel composition and returns a global location where only the relevant automata locations appear.

$$\begin{array}{lll} P_{Rel} & : (n+1) - \text{tuple de S} \rightarrow & (n+1) - \text{tuple de S} \cup \{\perp\} \\ P_{Rel} & (< l, a_1, \ldots, a_n >) = & < l, b_1, \ldots, b_n > / \\ & & b_i = \begin{cases} a_i \text{ si } A_i \in Rel(l) \\ \perp \text{ en caso contrario} \end{cases} \end{array}$$

Now, we can define the quotient automaton according to *Rel* as follows:

$$A_0 \parallel^{Rel} |A_1| ... |A_n = G_r = (S_r, X_r, \Sigma_r, A_r, I_r, s_{0_r})$$

where:

$$S_r = \{P_{Rel}(s) \in \bigotimes S_i \cup \{\perp\} / \exists s \in Locs(A_0|A_1|\ldots|A_n)\}$$

$$X_r = Clocks(X)$$

$$\Sigma_r = \bigcup_{i=0}^{n} Labels(A_i)$$

$$I_r(s_r) = \bigwedge_{\{i:0..n/i \in Rel(\Pi_0(s))\}} I_{A_i}(s_r)$$

$$s_{0_r} = P_{Rel}(Init(A_0|A_1|...|A_n))$$

$$\begin{aligned} A_r = \{ \; < P_{Rel}(s), a_r, \psi_r, \alpha_r, P_{Rel}(s') > / \;\; & \exists \; < s, a_r, \psi, \alpha, s' > \in Edges(A_0|A_1|\ldots|A_n) \wedge \\ & \exists i \in Rel(\Pi_0(s)) \wedge a_r \in Label(A_i) \wedge \\ & (\nexists i : 0..n)(i \notin Rel(\Pi_0(s)) \wedge e_r \in Output(A_i)) \\ & \wedge \psi_r = \bigwedge_{\{i:0..n/i \in Rel(\Pi_0(s))\}} \Pi_i(\psi) \\ & \wedge \alpha_r = \bigcup_{\{i:0..n/i \in Rel(\Pi_0(s))\}} \Pi_i(\alpha)\} \end{aligned}$$

Let us characterize the transitions of the quotient automata.

Fact 2 *Let $p, p' \in S_r$. Let $Rel_p = Rel(\Pi_0(p)^{@})$.*

- *Discrete Transitions*

$$a \in \bigcup_{0 \le i \le n : i \in Rel(\pi_0^{@}(p))} \Sigma_i \; \wedge \; (\nexists i : 0..n)(\Pi_i(p) = \perp \wedge e_r \in Output(A_i)) \wedge$$

$$\frac{\begin{array}{c}(\forall i)(0 \le i \le n : (a \notin \Sigma_i \vee i \notin Rel(\pi_0^{@}(p)) \wedge \pi_i(p) = \pi_i(p')) \vee \\ (a \in \Sigma_i \wedge \pi_i(p) \mapsto_0^a q \wedge (\pi_i(p') = q) \vee \pi_i(p') = \perp))\end{array}}{p \mapsto_0^a p'}$$

- *Temporal Transitions*

$$\frac{(\forall i) 0 \le i \le n : i \in Rel_p \Rightarrow \Pi_i(p) \mapsto_t^{\lambda} \Pi_i(p')}{p \mapsto_t^{\lambda} p'}$$

Example: 18 *The figure shows a normal product of TA and a quotient product. Shadows identify equivalence classes according to P_{Rel}; they are translated into one location in the quotient automata. Table 7.1 shows some of the calculated values to conclude the quotient. Note that $\neg(A_1 \xrightarrow{\times} A_0(S_0^*(1), S_0^*(0)))$, $\neg(A_2 \xrightarrow{\times} A_0(S_l^*(1), S_0^*(1)))$.*

Obs.Locs (q)	Rel(q)	$S_q^*(A_0)$	$S_q^*(A_1)$	$S_q^*(A_1)$
0	0,1,2	0	0	0
1	0,2	1	0,1	1
2	0	2	0,1	0,2
3	0	3	0,1	1

Table 7.1: Calculated Values

Algorithm to Calculate the Quotient Automata using *Rel* The following algorithm receives a generic *Rel* satisfying the second postulate and builds the reachable part of the quotient automata. It uses a simple graph traversal-building scheme keeping the already processed nodes and the ones to explore. That is, it is very similar to the algorithm that builds the standard composition. In our case, nodes represent equivalence classes, they are tuples with a bottom sign in the non-relevant components. For each node it is calculated all the enabled transitions as the definition states $(TransEnabled(s, A_0, \ldots, A_n) = \{t \in A_r / Src(t) = s)$. Let us assume that (a) components provide a label-ordered list of transitions in $O(1)$ for each location, and (b) the algorithm explores the labels following that order. Then, the algorithm executes $O(V + E)$ steps where V is the number of nodes and E is the number of edges of the final quotient graph. This is due to the fact that each node and transition is generated once, and $TransEnabled$ procedure only considers set of transitions that share the same label by applying a merge strategy to traverse the lists of transitions of each location of s.

It is not difficult to see that the quotient automaton can be easily build "on the fly", that is by demand by the verification engine like [61].

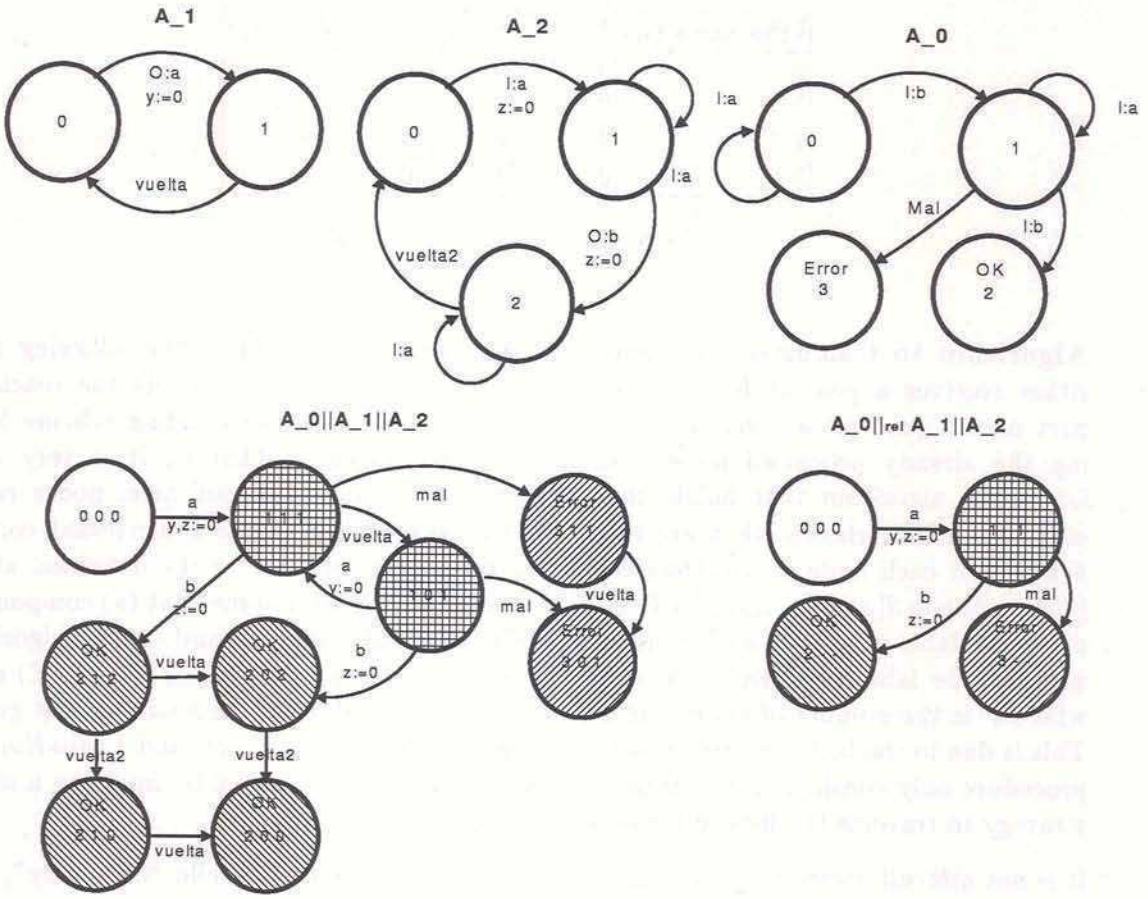


Figure 7.1: Standard vs. Reduced Composition

$\text{Compose}_{Rel}(A_0, A_1, ..., A_n, Rel)$ returns $< S_r, X_r, A_r, I_r, s_{0_r} >$

Let $s_(0_r)$ such that $\pi_i(s_{0_r}) = \begin{cases} \perp \text{ if } A_i \notin Rel(Init(A_0)) \\ Init(A_i) \text{ otherwise} \end{cases}$

$ToProcess := \{s_{0_r}\}$
While $\exists s \in ToProcess$
Takes
$S_r" := S_r \cup \{s\}$
$E = \text{TransEnabled}(s, A_0, A_1, ..., A_n)$
For each $t_e \in E$
Build $t_r =< s, Label(t_e), \psi_r, \alpha_r, s' >$ where
$\psi_r = \bigwedge_{\{i/\pi_i(t_e) \neq Null\}} Guard(\pi_i(t_e))$

$\alpha_r = \bigcup_{\{i/\pi_i(t_e)\neq Null\}} Reset(\pi_i(t_e))$

$$\pi_i(s') = \begin{cases} \bot \text{ si } Ai \notin Rel(\pi_{i+1}(s')) \\ tgt(\pi_i(t_e)) \text{ if } \pi_i(t_e) \neq Null \\ \pi_i(s) \text{ if } \pi_i(t_e) = Null \end{cases}$$

If $t_r \notin A$

$A_r := A_r \cup \{t_r\}$

If $s' \notin ToProcess$

$ToProcess = ToProcess \cup \{s'\}$

$S_r := S_r \cup \{s'\}$

End For

$I_r(s) := \bigwedge_{\{i:0..n/\pi_i(s)\neq\bot\}} I(A_i)(\pi_i(s))$

End While

$X_r := \bigcup_{i=0}^{n} Clocks(A_i)$

End Compose

TransEnabled$(s, A_0, \ldots, A_n)$ returns E = set of $(n+1)$-tuple of $(Edges \cup \{Null\})$

E:= $\emptyset$

Foreach Label a

If $(\forall\, i : 0..n/\pi_i(s) \neq \bot \Rightarrow T_i(a,s) \neq \emptyset \wedge\ (\pi_i(s) = \bot \wedge (a \in Labels(A_i) \Rightarrow a \notin Output(A_i)))$

then $E := \{< t_1, \ldots, t_n > /(\forall i : 0..n(t_i \in T_i(a,s)) \vee (T_i(a,s) = \emptyset \wedge t_i = Null))\} \cup E$

else $E := \emptyset$

Where $T_i(a,s) = \{e/e \in Edges(A_i) \wedge Label(e) = a \wedge src(e) = \pi_i(s)\}$

End TransEnabled

7.3 Results

Now we show that if we quotient according a relevance function satisfying the former postulates the quotient composition is CO-bisimilar wrt. the normal composition.

Theorem 8 *Given an indexed set of I/O Timed Components $\{A_i\}_{0\leq i\leq n}$, and an assignment mapping $\mathcal{P} : Props \mapsto 2^{Locs(A_0)}$, If Rel is a relevance function satisfying the postulates of Def .30 then $(A_0 \parallel^{Rel} \{A_i/1 \leq i \leq n\} \simeq^{\mathcal{P}'^*,\mathcal{P}^*} [(A_0 \parallel A_1 \parallel \ldots \parallel A_n)]$ where $\mathcal{P}^*$ and $\mathcal{P}'^*$ are the natural extensions of $\mathcal{P}$ on the locations of $[A_0 \parallel A_1 \parallel \ldots \parallel A_n]$ and $(A_0 \parallel^{Rel} \{A_i/1 \leq i \leq n\})$ resp.*

Proof 16 *Let $A = [(A_0 \parallel A_1 \parallel \ldots \parallel A_n)]$ and $A' = (A_0 \parallel^{Rel} \{A_i/1 \leq i \leq n\})$. Now, we define the symmetrical relation that is our candidate bisimulation: $(p,q) \in B$ iff $(p \in G_A, q \in G_{A'}$ and $\forall 0 \leq i \leq n : @\Pi_i(q) = \bot \vee \Pi_i(p) = \Pi_i(q))$ or $(p \in G_{A'}, q \in G_A$ and $\forall 0 \leq i \leq n : @\Pi_i(p) = \bot \vee \Pi_i(p) = \Pi_i(q))$.*

Let us start analyzing the case $p \in G_{A'}, q \in G_A$. Note that since A_0 is always relevant (Item I of Def. 30) and thus $\Pi_0(p) = \Pi_0(q)$, then $[p]^*_{\mathcal{P}} = [q]^*_{\mathcal{P}}$

Case I: $p \mapsto_0^a p'$.

(a) If no irrelevant component exports the label "a" then, clearly $q \mapsto_0^a q'$ *with* $(p', q') \in B$ *In fact, the same discrete transition is taken by the relevant part while the irrelevant components stay in the same state local state. Note that Item II of Def. 30 is needed to know that the relevance function at a possible new* A_0 *location does not incorporate any new component wrt. to the source location.*

(b) An irrelevant component exports the label "a" as output label. This is not possible due to the definition of the quotient (Fact 2).

(c) All irrelevant components export the label "a" as input selections of size 1. Then, they are always enabled and $q \mapsto_0^a q'$ *with* $(p', q') \in B$. *In fact, the same discrete transition is taken by the relevant part while the irrelevant components perform the jumps dictated by that input. Note that Item II is needed to know that the relevance function at a possible new* A_0 *location does not incorporate any new component wrt. to the source location.*

(d) There is one irrelevant component A_k *exporting "a" as part of an input selection I of size greater than one. Since it is irrelevant we know that all relevant components exporting "a" can perform the same change of state with any other "a"'* $\in I$ *actually enabled in* $\Pi_k(q)$, *therefore* $q \mapsto_0^{a'} q'$ *with* $(p', q') \in B$ *(again using Item II).*

Note that in all cases I the $q \overset{B,p,\mathcal{P}'^* \cup \mathcal{P}^*}{\longrightarrow}_0 q'$ *part of the defintion of CO-bisimulation is a stutter.*

Case II: $p \mapsto_t^\lambda p'$ *and* $t > 0$ *Like from p, from p' the relevant part can advance in t time units. On the other hand, due to component definition, we can make the irrelevant part advance t time units avoiding input transitions (and also selecting the output accordingly to the state of the relevant part). This correction can be done due to non-transientness of output. Also, by definition of components, output and internal transitions cannot be blocked by the relevant part. Moreover, any discrete output transition of the irrelevant TA* A_i *does not change the* A_0 *location l by postulate I and does not influence (i.e. does not change the location or reset clocks) of relevant TA* A_k *by postulate III (i.e.* $(A_i \xrightarrow{\times} A_k(S_l(i), S_l(k)))$*). Therefore, in every position of that run the corresponding state is CO-bisimilar to the one obtained from p delaying the time of that position. Then a state* q' *is reachable from q such that* $(q, q') \in B$ *(the relevance function remains the same).*

The proof for the case $p \in G_A$ *and* $q \in G_{A'}$ *is easier.* $A' = (A_0 \parallel^{Rel} \{A_i / 1 \leq i \leq n\}$ *is less constrained than the original system A (except for the outputs that cannot be performed). Lets suppose that* $p \mapsto^a p'$ *and p and p' differ in the relevant part (the other case is trivially solved by stuttering). Then, "a" is a label exported by a relevant component. That, a-transition is enabled in all relevant components except the case that a is an output label of a irrelevant component (see Fact 2). However, if* A_k *exports "a" as output then by postulates III and I it necessarily belongs to the relevant set (since p' differs from p in the relevant part, it must perform a non-stutter input in a relevant component). Therefore,* $q \mapsto^{a'} q'$ *and* $(p', q') \in B$ *(due to postulate II). The time transition is straightforward due*

to the fact that the system is less or equally constrained (see Fact 2).

It is clear that the continuos observational bisimulation between both LTS shown here respects the propositional assignment (moreover, the A_0 location and clocks are the same in observational timed bisimilar states).

From Theorem 3 we obtain:

Corollary 6 *Given a TA A_0 and an indexed set of components $\{A_i\}_{1\leq i\leq n}$, an assignment mapping $\mathcal{P} : Props \mapsto 2^{Locs(A_0)}$, Rel is a relevant function satisfying the postulates; let $\mathcal{P}^*$ and $\mathcal{P}'^*$ be the natural extensions of $\mathcal{P}$ on the locations of $[A_0 \parallel A_1 \parallel ... \parallel A_n]$ and $(A_0 \parallel^{Rel} \{A_i/1 \leq i \leq n\})$ resp. Then, for all TCTL formula ϕ then $(A_0 \parallel^{Rel} \{A_i/1 \leq i \leq n\}) \models_{\mathcal{P}'^*} \phi \iff [A_0 \parallel A_1 \parallel ... \parallel A_n] \models_{\mathcal{P}^*} \phi$.*

Corollary 7 *The local reachability problem on A_0 for $(A_0 \parallel^{Rel} \{A_i/1 \leq i \leq n\})$ is equivalent in $[(A_0 \parallel A_1 \parallel ... \parallel A_n)]$.*

The proof is trivial. Reachability can be written as a TCTL formula.

Also note that if *Rel* does not satisfy the stated properties then the method is still conservative.

7.4 Examples

To validate the potential of this technique we develop a prototype preprocessing tool based on these ideas [74]. We present several examples to illustrate the reduction method. We vary the value of the constant in the observers to get cases where the error location is reachable and cases where it is unreachable. All cases were checked using backward analysis while reachable ones are also checked using forward analysis [61]. The backward verification column is measured in seconds while the forward information is given in terms of symbolic states and transitions. We applied KRONOS tool [61] (version 2.4.3) on a Windows 98 platform (Pentium III 400Mhz, 64 MB). In tables, bottom symbol ($\perp$) means that the verification tool has not finished after 10hs of processing for the backward case or after 100.000 symbolic states were generated in forward mode.

RCS Let us start applying the method to the Rail Crossing Example presented in Sect. 3.3. We want to check that it is not possible that the train 1 traverses the rail cross when the gate is not completely down or very soon after it has gone down. Figure 7.2 shows a safety observer that follows the expected behavior of the system. That is, after the gate is risen a lower signal shall be issued and the gate shall go down before the train enters the rail cross ($in1$ event), then the train can not traverse the rail cross before 2 time units.

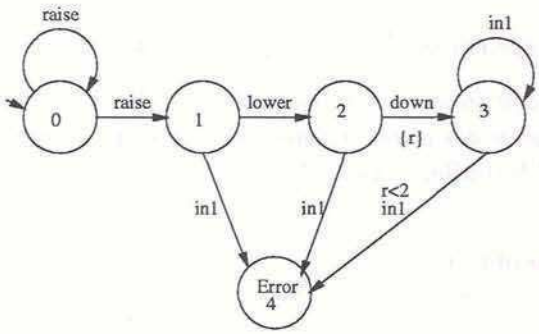


Figure 7.2: Observer for Rail Cross System

If we intuitively calculate the relevance of components we can conclude that the controller is no longer needed when the observer is at location number 2 (and therefore no other train but train 1). At location number 3 no component except the Train 1 is relevant. At location 4, a trap location, no component is relevant (see Fig 7.2). The Table 7.3 compares the sizes of the standard and the quotient system for the system with four trains.

Obs.Locs.	Relevants
0	$Obs, Train1, Train2, Train3, Train4, Gate, Controler$
1	$Obs, Train1, Train2, Train3, Train4, Gate, Controler$
2	$Obs, Train1, Gate$
3	$Obs, Train1$
4	Obs

Table 7.2: Relevance Function

Composition	#Locs.	#Edges	Reach False	Reach True	
			Backward	Backward	Forward
Standard	1026	5180	41.19s	170.45s	26203 st. 30438t
Quotient	437	2218	1.57s	160.42s	26176 st. 30402t

Table 7.3: Standard Composition vs. Quotient for Rail Crossing System n=4

Fault Detection Now, we apply the technique to the Fault Detection Net requirement for the Mine Pump (see Fig. 7.4), that is if a fault on the main processor or the net must be informed within 2 seconds to the remote Operator.

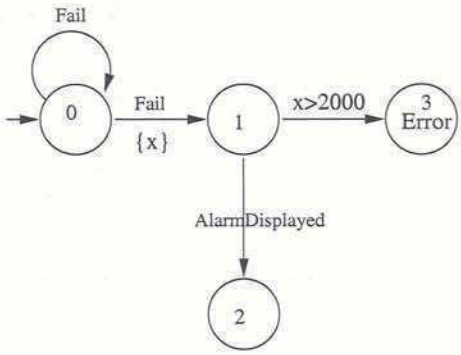


Figure 7.3: Fault Detection Net Observer

Recall that the relevance calculus detects that components involved in the query are: the Console Proxy, Net, Console Displayer, Timer, and the Display WatchDog. In Table 7.4 it is depicted the relevance function while in Table 7.5 we show the sizes and verification times for this example.

Observer Loc.	Relevants
0	$Obs, Cons.Proxy, Net, ConsoleDisplayer, Timer, WatchDog$
1	$Obs, ConsoleDisplayer, Timer, WatchDog$
2	Obs
3	Obs

Table 7.4: Relevance Function for Fault Detection Net

Composition	#Locs.	#Edges	Reach False	Reach True	
			Backward	Backward	Forward
Loop-Back, Standard over Relevant Components	225	1050	4634.91s	1410.35s	126 st. 132 t.
Trap, Standard over Relevant Components	270	1206	4632.36s	1412.04s	126 st. 132 t.
Trap, Quotient over Relevant Components	152	745	0.72s	21.10s	6 st. 5 t.

Table 7.5: Standard Composition vs. Quotient for FaultDetection NET Observer

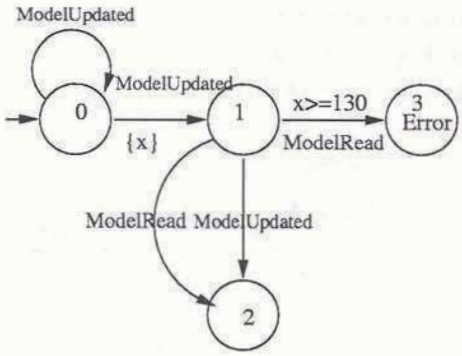


Figure 7.4: Freshness Observer

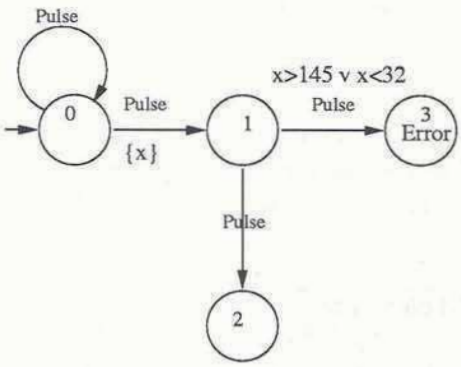


Figure 7.5: Regularity Observer

Active Structural The second example shows how the reduction method works in the requirements for the design of the Active Structure Control System. Even if the preprocessing method of presented in Def. 26 is not applied the calculus of the relevance function presented in this chapter discovers the same irrelevant TA. In fact, in the case of the freshness observer this method discovers the irrelevance of the actuator and the component that models the communication with the pulser task. Something similar occurs with the distance query where the calculus detects the irrelevance of the modeler task, the sensor, and the component modeling their communication. Table 7.7 shows the intractability of the standard compositions over all TA. We also present an standard parallel composition of the relevant components as calculated by the relevance calculus (see Def. 26).

For the Active Structural System, we add a new requirement to further illustrate the power of the quotient method. In this case we want to check the age of the pulse wrt. to the time when the model was updated with the value used to calculate the pulse magnitude. We call this requirement *Pulse Freshness*. Two possible safety observer for this requirements

Observer Loc.	Relevants
0	$Obs, Sensor, Comm, Modeler, Pulser$
1	$Obs, Sensor, Comm, Modeler, Pulser$
2	Obs
3	Obs

Table 7.6: Relevance Function for Freshness

Composition	#Locs.	#Edges	Reach False	Reach True	
			Backward	Backward	Forward
Loop-back, standard with all TA	768	2720	$\perp$	$\perp$	37391 st. 45013 t.
Trap, standard with all TA	1008	3530	$\perp$	$\perp$	37391 st. 45013 t.
Trap, Standard over Relevant Components	126	331	0.90s	12.37s	644 st. 699 t.
Trap, Quotient over Relevant Components	62	161	0.44s	11.28s	414 st. 459 t.

Table 7.7: Standard Composition Vs Quotient for Freshness

can be seen in Fig. 7.6. The second one is a more detailed version that may illustrate how adding detail to the observed sequence may lead to greater reduction. The tables show a dramatic time reduction.

Observer Loc.	Relevants
0	$Obs, Pulser, Comm1, Actuator$
1	$Obs, Pulser, Comm1, Actuator$
2	Obs
3	Obs

Table 7.8: Relevance Function for Regularity

Composition	#Locs.	#Edges	Reach False	Reach True	
			Backward	Backward	Forward
Loop-Back, standard over Relevant Components	72	153	0.20s	0.09s	38 st. 38 t.
Trap, standard over Relevant Components	96	198	0.07s	0.07s	38 st. 38 t.
Trap, quotient over all TA	45	87	0.05s	0.05s	38 st. 38 t.

Table 7.9: Standard Composition Vs Quotient for Regularity

CSMA To further illustrate the generality of the method, the method is applied to another generic compositions of I/O components (it is not real-time system design models as the previous two). This is the case of the CSMA/CD example (Sect.3.3). We want to check the bounded delay for collision detection [128]. Unlike that presentation we model the property by means of an observer automaton that would detect the case where a station messages collide and the collision detection is not received before 26 ms and the end events (Fig. 7.7).[3]

7.5 Conclusions and Discussions

We will try to answer some questions and guesses about the effectiveness of our method, namely:

- the less coupled the system the more reduction is obtained,
- it is better to build safety observers with trap locations instead of observers that loop back to initial location when the sequence of events is not a counterexample, etc.
- is it possible to treat examples that are intractable with the standard composition?.

Graphically, the technique achieves interesting reductions when the system is "loosely coupled" like the model build from RTS Designs. We say that a model is loosely coupled when strongly connected components of a graph which nodes are the I/O timed components

[3]In [128], the TCTL formula $init \rightarrow (\forall 2 TRANS1 \wedge TRANS2 \rightarrow \forall \diamond_{\leq 26} RETRY1)$ is used instead. In our test bed, the verification of this formula took 56.5 sec over the system under analysis which size is 344 Locs. and 2272 Trans.

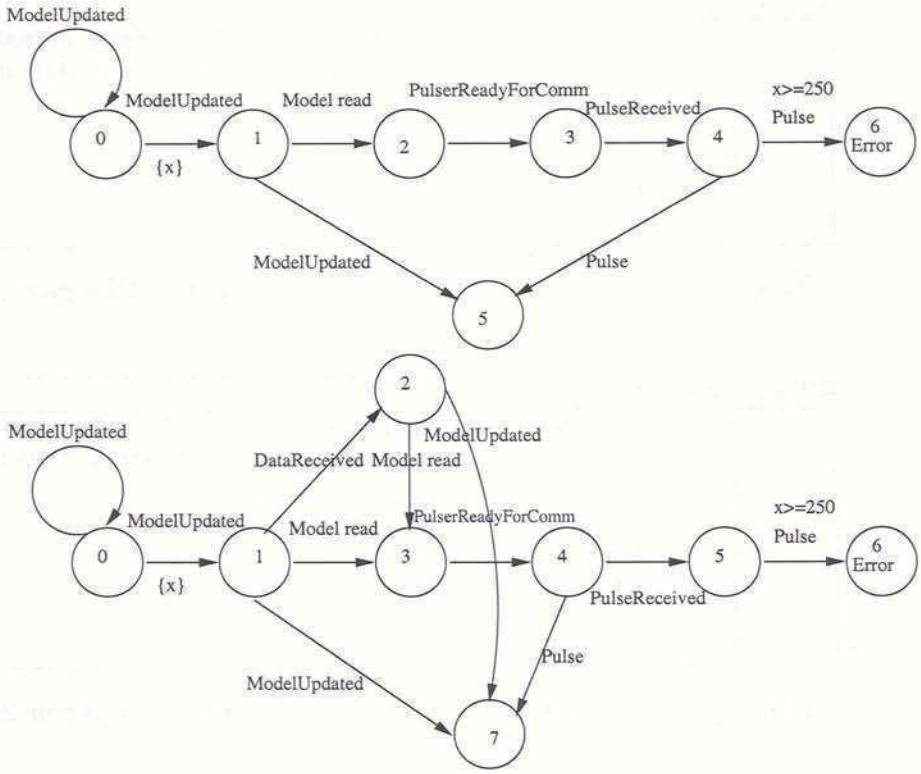


Figure 7.6: Two Safety Observers for the Pulse Freshness Requirement

and the edges are the influence relation between components are small. However, examples like the Rail Cross System and the CSMA/CD, which are rather standard and coupled examples, the technique still makes some reductions. This is due to the fact that at some locations of the observer few components have real influence on the future behavior.

Observers generally choose non-deterministically when to start a sequence of events that might exhibit a counterexample. When this is not the case, the observers can be built either to loop-back to the initial location or to enter into a trap location. Tables reveal that, despite the composition size, observers with trap locations are analyzed faster than observers that loops back when an event shows that the non-deterministic selection does no found a counterexample. Observers with trap locations are the ones used as the base of quotient compositions (note that a DAG topology is better for Item II of Def. 30 since all locations of a Strongly Connected Component of the observer must share the same relevant I/O timed Components).

In the example of 7.12 it is shown that sometimes the more detailed the observer is (in the sense of the sequence of events) the more reduction is obtained. This is the case when the detail shows an expected sequence of events that may serve to detect the irrelevance

Observer Loc.	Relevants
0	$Obs, Sensor, Comm, Modeler, Pulser, Comm1, Actuator$
1	$Obs, Sensor, Comm, Modeler, Pulser, Comm1, Actuator$
2	$Obs, Pulser, Comm1, Actuator$
3	$Obs, Comm1, Actuator$
4	$Obs, Actuator$
5	Obs
6	Obs

Table 7.10: Relevance Function for Pulse Freshness (Observer 1)

Observer Loc.	Relevants
0	$Obs, Sensor, Comm, Modeler, Pulser, Comm1, Actuator$
1	$Obs, Sensor, Comm, Modeler, Pulser, Comm1, Actuator$
2	$Obs, Modeler, Pulser, Comm1, Actuator$
3	$Obs, Pulser, Comm1, Actuator$
4	$Obs, Comm1, Actuator$
5	$Obs, Actuator$
6	Obs
7	Obs

Table 7.11: Relevance Function for Pulse Freshness (Observer 2)

of some I/O components.

It is worth noting that in some cases we were able to treat cases where the original standard composition was not possible to analyze (more than 10hs of processing). There are several cases (in general medium-size to big-size examples) where the time reduction was quite dramatic. That is, in several cases, while the number of locations and transitions was halved, the verification time was reduced in a much greater factor (2 to 6000 times faster!), specially when backwards analysis is performed on a correct system (i.e., when an error location is not reachable). The time-savings when analyzing systems where the error is reachable are more modest but still interesting (counterexamples are smaller in forward analysis). We still do not have a clear explanation of such a difference in effectivity of the method in the reachable and not reachable cases.[4] We did another experiment changing the constant appearing in an observer to relate the backwards verification times on the standard and the quotient composition. This was the case of the RCS observer(Fig. 7.2), where the constant of "train arrives too soon after gate comes down" was varied in the range 3 to 9. In all those cases the error is reachable. The table 7.15 shows that while the verification time on the quotient keeps more or less constant the verification effort on the standard composition explodes when the constant C is greater than 6 (in the guards,

[4]We guess that sometimes it might be the case that the counterexample is found without traversing too much of the irrelevant part of the graph. We have no clear guess about the way the backwards analysis behaves when the error is reachable.

Composition	#Locs.	#Edges	Reach False	Reach True	
			Backward	Backward	Forward
Trap, Standard over all TA	1268	4554	$\perp$	$\perp$	6670 st. 9253 t.
Trap, Quotient over all TA (observer 1)	543	1856	6341.31s	$\perp$	1478 st. 1890 t.
Trap, Quotient over all TA (observer 2)	495	1665	6140.94s	$\perp$	1478 st. 1820 t.

Table 7.12: Standard Composition Vs Quotient for Pulse Freshness

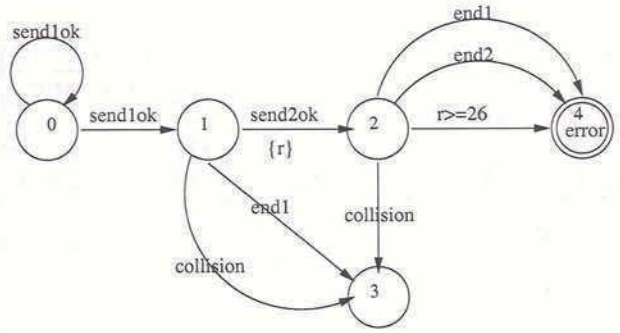


Figure 7.7: Collision Detection Observer for the CSMA/CD Protocol

$T1 > 6$ and $r > C$ appears always together). A reason that might explain that behavior is fact that in the quotient construction there is just one edge which guard depends on the observer clock r while in the standard construction the number of such edges is 108. We believe that this might become another source of time-savings of the method.

Note also that there is always size reduction in the trap locations of the observer. Those reductions on the size do not immediately imply an important reduction on analysis time when systems are analyzed using backward algorithms.[5]

Besides, when a component becomes irrelevant its clocks become "not active" according to the definition of [63]. This might lead to further and important reduction in analysis time if activity of clock were taken into account by the verification engine.

Finally, let us draw some general conclusions. This is a rather orthogonal technique to reduce two parameters of model complexity: number of locations and edges. It can be fed naively with the components of the system under analysis, and an observer; it discovers the underlying dependence among components to finally discard the components that do not influence the behavior up to the observational point of view. It always gets size reductions that, sometimes, imply significant to dramatic time savings during the verification step. This is generally the case when it is applied to medium-size to big-size examples (it could even make them treatable). It is rather easy to use and integrate as a preprocessor for known tools, it just requires the user to declare the I/O interface of TA. Last but not least,

[5]However, we believe there may be a significant time reduction when the system is analyzed forward and the error location is not reachable. Our guess is based on the fact that all the global locations that are locally a trap for the observer in the standard composition are reduced into one location in the quotient.

Observer Loc.	Relevants
0	$Obs, Bus, Sender1, Sender2, Sender3, Sender4, Sender5$
1	$Obs, Bus, Sender1, Sender2, Sender3, Sender4, Sender5$
2	$Obs, Bus, Sender1, Sender2$
3	Obs
4	Obs

Table 7.13: Relevance Function for Collision Detection

Composition	#Locs.	#Edges	Reach False	Reach True	
			Backward	Backward	Forward
Loop-Back, Standard over all TA	811	5605	3.082s	365.75s	71 st. 71 t.
Trap, Standard over all TA	1155	7877	3.079s	301.48s	71 st. 70 t.
Trap, Quotient over all TA	443	3148	0.324s	96.09s	66 st. 65 t.

Table 7.14: Standard Composition vs Quotient for Collision Detection

this technique could be applied to untimed systems, they are special cases of these timed systems [6]

[6] $S_l(i)$ sets can be overestimated without calculating the whole Cartesian product, like pair wise composition ,i.e., procedure (b).

Constant Value C	Verif. Time on Standard	Verif. Time on Quotient
3	186.92s	158.36s
4	165.97s	156.45s
5	177.04s	163.33s
6	176.42s	159.73s
7	823.57s	159.86s
8	1084.75s	160.06s
10	¿2000s	164.02s

Table 7.15: Varying the Constants for the RCS Example

Chapter 8

Fitting the Pieces Together

8.1 An Architecture for the Checking Tool

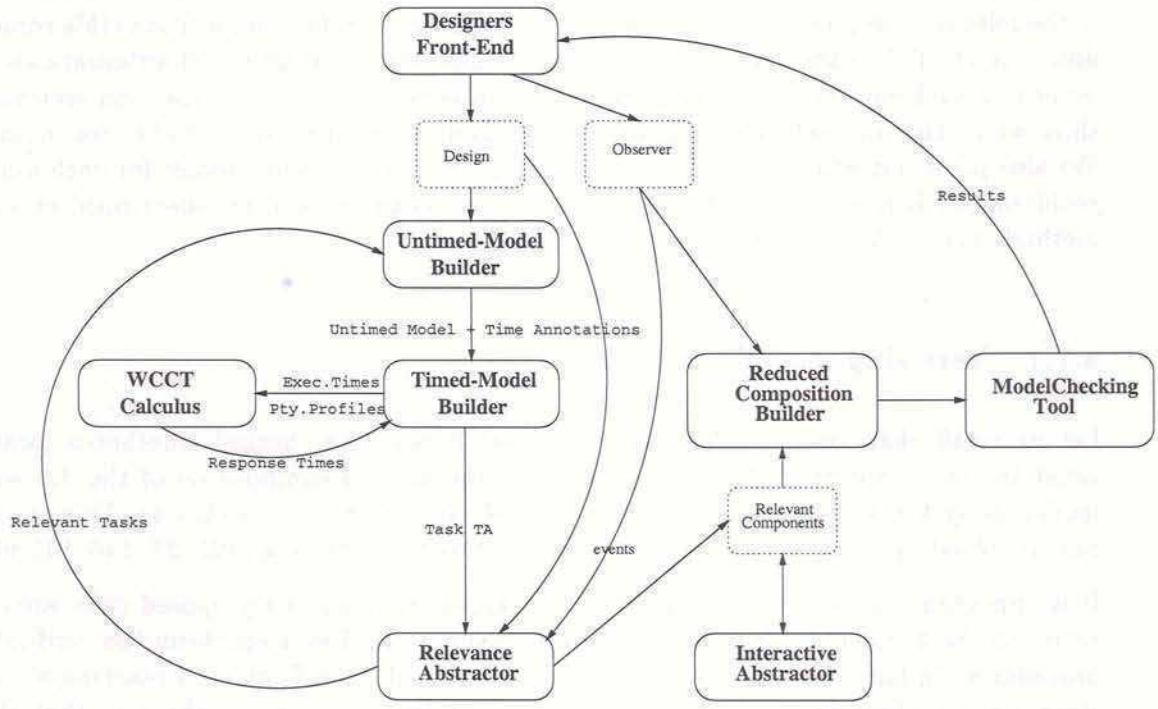


Figure 8.1: The Tool Architecture

We build a prototype to validate our technique, and currently, we are working on a deliverable version of the tool. We believe that it is worth describing the conceptual architecture of that tool to understand how the modeling, reduction and checking methods are effectively integrated. We also present some performance results that show the feasibility of

our approach.

The architecture is depicted in Fig. 8.1. Designers use *Designers Front-End* to describe the physical design along with observers for the scenarios associated with properties and requirements (Chapter. 4).

Given the events involved in a particular observer, the *Relevance Abstractor* determines the components and the level of abstraction actually required to perform the verification (see Sect. 6.1). The relevance abstractor has three functionalities: (a) it detects the Relevant Components and the Relevant Events (Def. 26), (b) it communicates the relevant tasks to be translated to the model-builder subsystem, and (c) it reduces the size of the resulting tasks models at the level of abstraction dictated by the relevant events (Def. 27). The *Untimed-Model Builder* produces the untimed model as explained in Sect.5.1. Then, the *Timed-Model Builder* produces the temporization by calculating best and worst case response times (Sect. 5.2).[1]

Also, an *Interactive Abstractor* is available to further manipulate the model by applying conservative rules (Sect. 6.2).

The preprocessing algorithm presented in chapter 7 is feed with the timed-automata model of the relevant components further reduce the size of the parallel composition (this could be done "on the fly" in the verification engine). Finally, the resulting timed-automata are the input of a back-end model-checking tool which provides the results. In the next section, we show which kind of model-checking technology/tool is needed for each kind of requirement. We also point out which reduction techniques produce equivalent models for each kind of requirement. It is worth saying that we will resort to existing and well-developed checking methods and tools whenever they are available.

8.1.1 Verifying Safety Observers

Let us recall that for the safety observers case it must be checked whether a location satisfying the proposition *Error* is reachable in the parallel composition of the TA which model the system under analysis *SUA* and the observer automaton (Sect. 4.4). This problem can be solved by using any tool that supports TA verification (e.g., [61, 23, 146, 91], etc.).

It is important to remark that the models obtained following the proposed rules are non-zeno, (see Sect. 3.3.1); which is a key property usually required to perform the verification procedures. In fact, if non-zenoness were not guaranteed, then finite runs reaching a "bad" state are not necessarily witnesses of reachability since it might be the case that those finite runs cannot be extended into a time-divergent runs (a necessary condition for a run to belong to the automaton semantics). Moreover, zenoness may "stop" time progress and an error may remain undetected.

[1] Note that this is just one possible solution. For example, the whole model could be translated to the maximum detail. The abstract submodels for each requirement could be built afterwards.

Requirement	Result	Exec. (Secs.)
Freshness	OK	0.44
Regularity	OK	0.19

Table 8.1: Results of the Queries: Active Structural System

Requirement	Result	Exec. (secs)
Separation	OK	0.005
Response 1	OK	0.175
Response 2	OK	0.074
Bounded 3.0	OK	0.025
Bounded 3.1	OK	13.005
Freshness 1	ERROR	0.015
Correlation	OK	0.109
False Alarm CO	OK	0.009
Fault Detection HLW	OK	0.865
False Alarm HLW	OK	0.17
Fault Detection NET	OK	0.72
Freshness 2	ERROR	0.069
Console	ERROR	1.635

Table 8.2: Results of the Queries: Mine Pump System

As was remarked in Sect. 6, given a system under analysis SUA, and an observer O with a labeling $\mathcal{P}$; $[SUA] \approx^{RE} [RC_{RE}]$ and $Labels(O) \subseteq RE$ then, by Corollary 1 we got $[SUA] \parallel [O] \approx^{\mathcal{P}_1,\mathcal{P}_2} [RC_{RE}] \parallel [O] = [RC_{RE} \parallel O]$ where $\mathcal{P}_i (i = 1..2)$ are the natural extensions of $\mathcal{P}$ to the respective cartesian products. Thus by Theorem 1 the error location is reachable in $[SUA \parallel O]$ if and only if it is reachable in $[RC_{RE} \parallel O]$. Moreover, we can get a further reduction: by Theorem 8 $(O \parallel^{Rel} RC_{RE}) \simeq^{\mathcal{P}_2',\mathcal{P}_2} [RC_{RE} \parallel O]$. Thanks to the fact that CO-bismulation implies simulation, transitivity of simulation, we got : $(O \parallel^{Rel} RC_{RE} \approx^{\mathcal{P}_2',\mathcal{P}_1} [SUA \parallel O]$. Finally, by Theorem 1, we can conclude that the original reachability problem in $[SUA \parallel O]$ is the same than the reachability problem in $O \parallel^{Rel} RC_{RE}$, that is the problem in the quotient composition presented in Chapter 7. Thus, all the so claimed exact abstraction techniques we develop can be applied to reduce the model without sacrificing accuracy.

We applied KRONOS tool [61] (version 2.4.3) on a Windows 98 platform (Pentium III 400Mhz, 64 Mbytes) to check the requirements. We run the backwards exploration option since the reachability option (forward exploration) showed a rather irregular pattern of execution times on the examples. We would use the reachability option when it is likely to have a counterexample. The results are summarized in Table 8.1.1 and seem really

promissory.

For the requirements not met by the proposed design, we have easily discovered - by experimenting with different parameter values - the properties which are actually satisfied by the design. Some of these values might be acceptable while others might imply a redesign guided by the counterexamples provided by the analysis tool.

- The CO readings are at most 130 ms old.
- The age difference for CO and CH4 values paired in the logger is at most 113 ms.
- The CH4 value at the CH4-status Object is not older than 180 ms.
- There are at most five alarms informed by the CH4 sensor within two "get package" operations.

8.1.2 Verifying Büchi Observers

It is easy to see that:

Lemma 9 *Let $B = (O, F)$ a Büchi observer for A then $A \times B = ([A \parallel O], \{(s, f) \in Locs(A) \times Locs(O)/f \in F\})$ is a Büchi TA such that $L^\infty(A) \cap L^\infty(B) \neq \emptyset$ iff $L^\infty(A \times B)$ is non-empty.*

To solve that problem for strongly non-zeno TA A we can resort to the time abstracting technique presented in [149] for Timed Büchi Automata model checking. By definition of components, every infinite run of a closed system -that is a component with no input label- is time divergent. In fact, infinite runs with no-input labels must necessarily diverge. This hypothesis is enough use the techniques presented in [149]. That is, due to lemma 9 we what to check whether the Büchi TA $A \times B$ has non-empty language. Given a Strong Time-abstraction Bisimulation $\sim$ on $[A \parallel O]$ such that $q \sim p$ then $\Pi_O(p)^@ \in F$ iff $\Pi_O(q)^@ \in F$, let G the $\sim$-quotient of $[A \parallel O]$. A node of G (i.e. a class) is called repeating if it contains only repeating states (i.e. states which O-projection of its location is in F). Note that if a node is not repeating then it contains no repeating states, by the fact that $\sim$ respects the repeating states. Then, we have the following result from [149]:

Lemma 10 *$A \times B$ has non-empty language if and only if G has a maximal Strongly Connected Component containing a repeating node.*

In [149] it is shown how to solve this problem.

Theorem 9 *Given two components A_1, A_2 and an observer O for both of them such that $[A_1] \simeq_{Label(O)} [A_2]$ then $L^\infty(O) \cap L^\infty(A_1) = \emptyset$ iff $L^\infty(O) \cap L^\infty(A_2) = \emptyset$.*

Proof 17 *Trivial since simulation equivalence means that the timed label-sequences are the same.*

This means that, like the case of the safety observers, all the exact abstraction techniques presented so far accurately preserve Büchi problem (Sect. 6.1 and Chapter 7).

8.1.3 Verifying TCTL Observers

TCTL requirements can be solved using model-checking tools like KRONOS [61].

As was shown, given a system under analysis SUA, and an observer O then $[SUA] \simeq^{RE} [RC]$ and $Labels(O) \subseteq RE$. Then by Corollary3, given a propositional assignment $\mathcal{P} : Props \mapsto 2^{Locs(O)}$, for all ϕ TCTL on $Props$, $[SUA \parallel O] \models_{\mathcal{P}_1} \phi$ iff $[RC \parallel O] \models_{\mathcal{P}_2} \phi$ where $\mathcal{P}_i (i = 1, 2)$ are the natural extensions of $\mathcal{P}$ to the respective cartesian products.

We can get further reduction. In fact, by Theorem 8 $(O \parallel^{Rel} RC \simeq^{\mathcal{P}_2', \mathcal{P}_2} [RC \parallel O]$, by transitivity of bisimulation, $(O \parallel^{Rel} RC \simeq^{\mathcal{P}_2', \mathcal{P}_1} [SUA \parallel O]$. Thus, by Theorem 3, we can conclude that $[SUA \parallel O]$ satisfies the same TCTL formulae than $O \parallel^{Rel} RC$.

8.1.4 Verifying Linear Duration Observers

Constraints on the accumulated time spent at particular system states are among the possible requirements for a real-time system. These requirements are called duration properties and, in general, they are can not be expressed by using previously presented types of observers. Besides, up to this thesis, there were no automatic technique that directly supports the verification of duration requirements over physical designs of real-time software.

To perform the verification of duration properties (actually, Linear Duration Invariants) over strongly non-zeno TA we present a new conservative technique in Chapter 9. Note that if we have a closed component, i.e. a component with no free input label, it must be a strongly non-zeno automaton. In fact, a cycle contains internal and/or output transitions and therefore any infinite run must be time-divergent.

As was shown, given a system under analysis SUA, and an observer O; $[SUA] \approx^{RE} [RC_{RE}]$ and $Labels(O) \subseteq RE$. Then, by Corollary 2, given a propositional assignment $\mathcal{P} : Props \mapsto 2^{Locs(O)}$ for all ϕ LDI, $[SUA \parallel O] \models_{\mathcal{P}_1} \phi$ iff $[RC_{RE} \parallel O] \models_{\mathcal{P}_2} \phi$ where $\mathcal{P}_i (i = 1, 2)$ are the natural extensions of $\mathcal{P}$ to the respective cartesian products.

Moreover, we can get a further reduction: by Theorem 8 $(O \parallel^{Rel} RC_{RE} \simeq^{\mathcal{P}_2', \mathcal{P}_2} [RC_{RE} \parallel O]$. Thanks to the fact that CO-bismulation implies simulation, transitivity of simulation, we got : $(O \parallel^{Rel} RC_{RE}) \approx^{\mathcal{P}_2', \mathcal{P}_1} [SUA \parallel O]$. Finally, by Theorem 2, we can conclude that the original reachability problem in $[SUA \parallel O]$ is the same than the LDI problem in $O \parallel^{Rel} RC_{RE}$, that is the problem in the quotient composition presented in Chapter 7. Thus, all the so claimed exact abstraction techniques we develop can be applied to reduce the model without scarifying accuracy.

8.2 Summary

To verify safety observers we just need to resort to a tool supporting reachability analysis of TA. Relevance calculus (Def. 26) (that is using only the detected Relevant Components), coarsening of the model (Def. 27), and quotient composition (Chapter 7) are exact abstraction techniques for these kind of requirements.

Büchi TA can be checked adapting the technique presented in [149] and supported by KRONOS. As in the case of safety observers, they are defined in terms of the linear time structure of the underlying LTS (indeed, safety observers are an special case of them). The same abstraction techniques are exact for them.

TCTL observers can be checked using tools like KRONOS [61]. Since in general is based on the branching structure of the underlying LTS the procedure for coarsening of the model (Def. 27) does not necessarily produce an exact abstraction. Fortunately, The relevance calculus (Def. 26), and the quotient composition (Chapter 7) are exact abstraction techniques for TCTL.

Duration Properties could be analyzed using the technique presented in Chapter 9. They are also based on the linear structure and all the abstraction techniques, namely relevance calculus (Def. 26), coarsening of the model (Def. 27), and quotient composition (Chapter 7) are exact abstraction techniques for these kind of requirements.

Chapter 9

Verifying Duration Properties

9.1 Introduction

Constraints on the accumulated sojourn time at particular system states are among the possible requirements for a real-time system. These requirements are called duration properties. In the previous chapter we see how Real-Time Systems Designs can be modeled by means of Timed Automata. We also point out that, unlike TCTL, there is no mature tool for verifying LDI (Linear Duration Invariants) over TA. This motivated our research on checking such kind of properties. Thus, in this chapter, we address the general problem of automatically verify whether a timed automata satisfies a duration property written as a Linear Duration Invariant. We extend a conservative algorithm presented in [28] which solves the problem using linear programming techniques. That is, we provide a procedure to translate timed automata into "timed" regular expressions for timed languages. Then, we apply a linear programming-based approach to this algebraic notation for the timed automata.

Our results in this chapter are more general than the ones presented in [154, 28]. Namely, TA is our starting point, and we can provide an accurate answer to the problem for a larger class of them.

9.1.1 Related Work

In [106] it is presented a computational model to identify a class of Hybrid Systems with decidable analysis problems: Integration Graphs. In that paper it is shown that reachability problem of integration graphs boils down into checking satisfability of some kind of duration properties (later called Linear Duration Invariants [161]) for finitary timed automata. Timed Automata [7] (TA) is one of the most widely used formalisms to model real-time systems. Linear Duration Invariants (LDI) are a fragment of Duration Calculus (DC) [160] used to express linear constraints on the accumulated time for the presence of

system states (see Chapter 2).

Thus, duration properties over Timed Automata allows to express more properties than standard reachability and they are still decidible [106]. In fact, runs that do not satisfy an accumulation relation can be detected or filtered out (e.g., "Does the Gas Burner system meet the requirement that leaking time should not be greater than the 5% of total time", "Does the protocol fail to meet a bounded response requirement under the Quality of Service assumption that the media guarantees that it fails no more than 1% of total time", etc.). The price one pays is the high complexity of the verification procedures. Also in [106] the authors present an algorithm for solving the problem satisfaction of an LDI by a Timed Automaton based on two techniques: digitization and mixed integer linear programming. Digitization is a way to obtain a discrete time automaton which generates the integer runs of the original dense-time version. Naive digitization is based on the region graph construction which produces huge graphs depending on the size of the constants involved in comparisons [7]. Besides, mixed integer linear programming techniques possess high complexity. An implementation for single clock automata is found in [60] (to treat composition of single clock automata some linear constraints are added to model synchronization.)

A related problem is treated in [6], the computation of accumulated delays. It does not cover the case of negative coefficients for durations and it is even more complex than the one presented in [106] for checking the "positive" LDI. In [85] a general approach for model-checking discrete Duration Calculus is presented. This approach is based on the inclusion of regular languages. Obtaining regular languages through digitization of TA as well as transforming DC formula into finite state automata are the main sources of its high complexity.

In order to obtain more practical algorithms, several proposals based on linear programming techniques were presented [161, 154, 64]. The common idea is to represent real-time systems by means of some sort of timed regular expression. These works are based on the fact that if the expression to verify is finite, i.e. there is no repetition in the expression, the verification can done by using a linear programming procedure (i.e., minimizing/maximizing the body of the LDI subject to the timing constraints in the form of linear inequalities on variables which stand for the duration of each locations visit). Therefore, most of the research effort is devoted to reduce the infinite case to the finite one, i.e. to eliminate repetition. The strength of these techniques is the reuse of well studied and efficient set of linear programming tools avoiding digitization. Their weakness lies on the fact that the starting points are those regular expressions, which can only represent a small class of TA.

However, in [28] we presented an algebraic formalism for expressing the behavior of the whole class of TA [1] along with a translation procedure. Based on the translation results, they provide a conservative analysis ("yes", "no", "don't know" answers) to the problem using linear programming techniques and avoiding digitization. Compared to the methods in [161, 154], the method presented in [28] and here can work directly on the whole class of

[1] In [9] is presented another algebraic formalism that is as expressive as TA.

TA and give the accurate answer to the problem for a larger subclass than any previously defined in these approaches.

Following these ideas, we also extend that approach to cope with a larger class of duration constraints. That is, we allow comparison using the $\leq, <, \geq$ *or* $>$ operands, while in [28] we only consider $\leq$. We therefore are able to check validity and satisfability questions for a duration property (i.e., Do all runs leading to a final location satisfy the duration property?, Is there a run leading to a final location that also satisfies the duration property?, resp.).

The chapter was organized as follows. In the next section, we present a working example. An algebraic formalism, Time Constrained Regular Expressions (TC-RE), for expressing the behavior of TA is given in sect.9.3 and the problem is translated to this formalism.

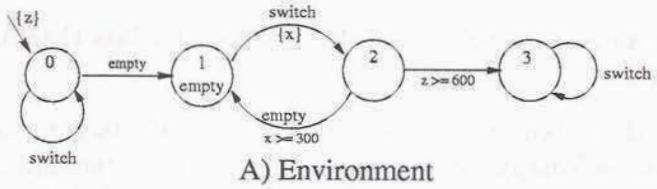
In section 9.5 the model-checking problem is solved for finite TC-RE (i.e., the ones generating a finite number of words). Then, the well-behaved TC-RE are presented in Sect. 9.6 which are TC-RE with good properties needed to deal with infinite TC-RE. Our model-checking procedure is finally presented in Sect. 9.8. Conclusions are summarized in the last section of the chapter.

9.2 A Case Study

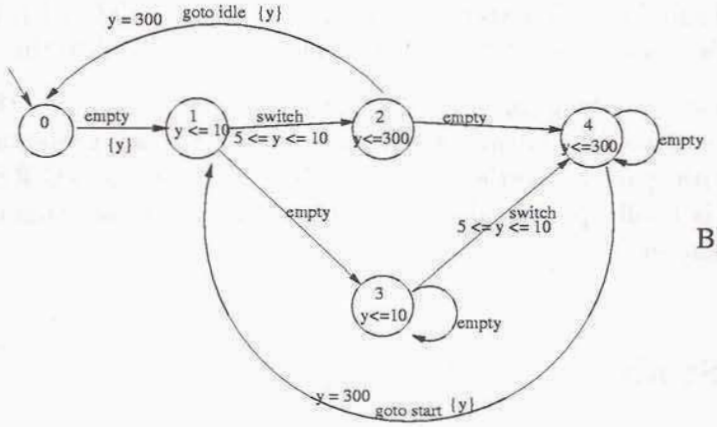
In this section, we sketch some elements of an embedded-software design and a duration requirement for a machine that pumps medicine into the patient blood. That machine should pump the right dosage at the right rate. Since the medicine should be pumped into the patient rather continuously, the machine accepts several vessels and switches, in a timely fashion, from an empty vessel to a full one [2] (and raises an alarm to indicate that the empty vessel should be replaced, if necessary). The fastest a vessel may run out of medicine is 300 t.u. The main "Quality of Service" requirement is that, provided there are always full vessels available, during a time interval longer than 600 t.u., the patient should receive the medicine at least 95% of the time.

The Physical Design The system design is composed of several tasks scheduled under fixed priority policy. The highest priority one is the *Switcher* task. As the name suggests, this task switches the machine from an empty vessel to a new one and then informs the event to a protected object *proxy console* which is eventually read by a task that sends the warning condition to a remote monitoring subsystem. It takes at most 5 t.u. to perform both subtasks in a dedicated environment. The Switcher task follows the sporadic server scheme [109] and has a minimum interarrival time of 300 t.u. That means that after serving an "empty" signal it is disabled until 300 t.u. have elapsed since the last signal arrival. However, in this case, it be can buffered up to one signal to be eventually served.

[2] The solution of filling a big vessel with the medicine was discarded due to safety and economical reasons. In fact, if the medicine is suspended the remaining content should be thrown away.



A) Environment



B) Switcher Task

Figure 9.1: Timed Automata Model

In Fig. 9.1 (a) is depicted the environment automaton with conveys the information about the minimal separation between two consecutive "empty" signals.

The Timed Automata Model and the Duration Property Although the Switcher task has the highest priority, it may be blocked by another client of the proxy console (e.g., the task that reads the alarm object and displays the alarm condition) due to PCP emulation [109]. Then, the WCCT calculus -see Sect. 5- returns 10 t.u. as the response time for the Switcher task. The TA shown in Fig. 9.1 (b) is the model produced by our technique for the Switcher task.

Formally, the duration requirement should be expressed over the parallel composition of the environment with whole model (which is not shown in this presentation). The duration property actually predicates on the projection of the environment state. Moreover, the abstraction method presented in Chapter 6 detects that only the Switcher task model affects the behavior of the the environment TA. Therefore, the requirement can be expressed as $\varphi = 20 \int Empty - \int true \leq 0$ with $F = \{(3,0),(3,1),(3,2),(3,3),(3,4)\}$ using the following propositional assignment over the parallel composition of the environment with the Switcher task model: $Props = \{Empty\}$ and $\mathcal{P} : Props \mapsto 2^{Locs(A)}$ be

$$\mathcal{P}(Empty) \;=\; \{(s_1, s_2) \;:\; s_1 = 1\}$$

The mapping α associated with the mapping $\mathcal{P}$ is

$$\alpha((s_1, s_2)) = \begin{cases} -1 & \text{if } s_1 \neq 1 \\ 19 & \text{if } s_1 = 1 \end{cases}$$

For example given $\sigma = (\perp, 00)\ (00, 11)\ (11, 22)\ (22, 20)$ and $\tau = (0\ 200\ 205\ 500)$, then $f^{\sigma}_{\varphi,\mathcal{P}}(\tau) =^{def} \sum_{1 \leq i < |\sigma|} \alpha(src(\sigma_i))(\tau_i - \tau_{i-1}) = (-1) \times 200 + 19 \times 5 + (-1) \times 295 = -400$.

9.3 Time Constrained Regular Expressions

In this section, we present a descriptive and algebraic representation of the finite behavior of TA: **timed constrained regular expressions** (TC-RE) [28]. TC-RE provides us the necessary insight to formulate the principles of the model-checking algorithm proposed later in this chapter. Hereafter, we will focus on TA where guards are conjunctions of atoms of the form : $x \sim c$ where x is a clock, $\sim \in \{=, \leq, \geq\}$ and $c \in I\!N$.[3]

Definition 31 (TC-RE) *A TC-RE is a tuple (R, Δ) where R is a regular expression over E_{id} and Δ is a finite set of triples of the form $(\delta, e, \sim c)$, where $\delta \subseteq E_{id}$, $e \in E_{id}$, $\sim \in \{<, >, \leq, \geq, =\}$ and $c \in I\!N$.*

The intuition behind this definition is that the regular expression (see [96]) R gives the potential untimed sequences of transitions (remember that E_{id} identifies edges, see Sect. 2.3.2), while Δ establishes a set of constraints on the distance between edge transitions. A tuple $(\delta, e, \sim c)$ in Δ is the analogous of a clock test associated with the edge e (which appears as the second component of the tuple). Then, the first component δ, hereafter called a *clock*, is the set of transitions which reset that clock. Roughly speaking, $\sim c$ is a time constraint on the distance between e and the closest previous δ-edge.

Definition 32 *Given a TC-RE (R, Δ), the language $L(R, \Delta)$ is the set of timed words (σ, τ) that satisfy:*

- *$\sigma \in R$ (i.e. σ is a transition sequence described by the regular expression),*
- *$\tau_0 = 0$ (initialization),*

[3]This constraint is quite reasonable for practical applications and simplifies the integer programming techniques which we are based on.

- $\forall\ 0 < i < |\sigma| : (\delta, \sigma_i, \sim c) \in \Delta \wedge \delta \cap \sigma_{i-1]} \neq \emptyset \Rightarrow \tau_i - \tau_{last\_\delta\_in\_\sigma_{i-1]}} \sim c$
  (i.e. τ satisfies the time constraints on the distance between edge instances).

Hereafter we write $\tau \in Sol(\sigma, \Delta)$ to express that (σ, τ) is a timed word satisfying the conditions of the last two items of Definition 32.

Definition 33 *Given an LDI formula φ:*

$$(R, \Delta) \models_{\mathcal{P}} \varphi \stackrel{def}{=} \forall (\sigma, \tau) \in L(R, \Delta) : (\sigma, \tau) \models_{\mathcal{P}} \varphi$$

Example: 19 *Let* $R = (\perp, 0)\ (0,1)((1,2)(2,1))^*(1,2)(2,3)(3,3)^*$ *and*

$$\Delta = \left\{ \begin{array}{lll} (\{(\perp,0)\}, & (2,3), & \geq 600), \\ (\{(1,2)\}, & (2,1), & \geq 300) \end{array} \right\}.$$

Then (R, Δ) is a TC-RE that represents all the timed words leading to the location 3 of the Environment automaton in Figure 9.1.

The notion of TC-RE leads to a clear separation between the untimed structure and the timing constraints. The structured nature of traditional Regular Expressions (RE) is extremely helpful for developing the principles of the algorithm. Given a timed automaton, it is easy to write down a TC-RE to express the timed language of the timed automaton (we will see in section 9.7).
Other kind of timed RE are also defined in [9], adding an intersection operator to define timed RE. As they proved, intersection is unavoidable with the "bound on length" style of timing constraints. Although their approach is elegant, we found rather difficult to extend the method we are proposing to this kind of timed regular expressions (in particular when intersection is combined with the Kleene closed operator). We avoid intersection making less explicit timing constraint constructs that are, at the same time, easier to manipulate.

The following facts are obvious from the above definition.

Fact 3 *Let R and R' be REs and let Δ be a set of constraints.*

1. *If R and R' are equivalent (i.e. they recognize the same language) then $L(R, \Delta) = L(R', \Delta)$.*
2. *If R recognizes a sub-language of R' then $L(R, \Delta) \subseteq L(R', \Delta)$.*

Fact 4 $\Delta \subseteq \Delta' \Rightarrow L(R, \Delta') \subseteq L(R, \Delta)$.

Definition 34 *The Prefixes of a Regular Expression R is the set of words $\{\theta / \exists \theta' : \theta\theta' \in R\}$.*

Definition 35 *A word σ over E_{id} is* **repeatable**, *denoted $Repeatable(\sigma)$, iff $src(first(\sigma)) = tgt(last(\sigma))$.*

It is obvious that

Fact 5 *$Repeatable(\sigma_{[i,j]})$ iff $src(\sigma_i) = tgt(\sigma_j)$.*

Definition 36 *Given a TC-RE (R, Δ), we define the timing for a clock δ in a run $\theta \in Prefixes(R)$, denoted $Times_\delta(\theta)$, as $\{last(\tau) - \tau_{last\_\delta\_in\_\theta} \mid \tau \in Sol(\theta, \Delta)\}$. It represents the possible final values of clock δ when timing is added to θ.*

Fact 6 *The set $Times_\delta(\theta)$ has minimum and, if it has an upper bound, then it has maximum too. We will note them resp. $MinTimes_\delta(\theta)$ and $MaxTimes_\delta(\theta)$, and we use $MaxTimes_\delta(\theta) = \infty$ if it has no upper bound.*

Definition 37 *Let (R, Δ) be a TC-RE. Let δ be a clock and $\theta \in Prefixes(R)$. Let MC be the maximum of the constants appearing in a time constraint in Δ. We define*

$$MIN_\delta(\theta) \stackrel{def}{=} \begin{cases} MinTimes_\delta(\theta) & \text{if } MinTimes_\delta(\theta) \leq MC \\ \infty & \text{otherwise} \end{cases}$$

and

$$MAX_\delta(\theta) \stackrel{def}{=} \begin{cases} MaxTimes_\delta(\theta) & \text{if } MaxTimes_\delta(\theta) \leq MC \\ \infty & \text{otherwise} \end{cases}$$

9.4 Problem Transformation in terms of TC-RE

To solve the validity problem $(A, F) \models \varphi$ we will show how to obtain a TC-RE capturing all the timed words of A leading to F. Then we only have to check if that TC-RE satisfies φ.

Given a timed automaton A and a set of final locations F, we could apply a simple procedure derived from the proof of Kleene theorem [96, 9] to (A, F) in order to obtain a regular expression that recognizes all the untimed words of transitions which lead to a final location in F.

On the other hand, Δ can be obtained simply as follows. For each edge $e =< s, a, \psi, \alpha, s' >$ follow this procedure:

- For each test $x \sim c \in \psi$ add the tuple $(\delta, < s, a, s' >, \sim c)$ to Δ, where δ is the set of transitions that reset the clock x (remember that this set includes also initial ones),
- For each test $x \sim c \in I(s)$ add the tuple $(\delta, < s, a, s' >, \sim c)$ to Δ, where δ is the set of transitions that reset the clock x as above.

We call the TC-RE constructed from (A, F) in this way $\Delta Kleene(A, F)$. Note that the second item could be eliminated if we assume that $\psi \Rightarrow I(s)$.

It is easy to see that the obtained TC-RE recognizes exactly the language of timed words of A leading to a final location in F. That is:

Fact 7 $(\sigma, \tau) \in L(\Delta Kleene(A, F))$ *iff* $(\sigma, \tau) \in L(A) \wedge tgt(last(\sigma)) \in F$.

Then using the properties of the automaton A we immediately get that

Theorem 10 *For a LDI formula* φ, $\Delta Kleene(A, F) \models_{\mathcal{P}} \varphi$ *iff* $(A, F) \models_{\mathcal{P}} \varphi$.

So from the automaton A we obtain a TC-RE (R, Δ), where R is a regular expression on the set of transitions E_{id} over the set of locations of the timed automata $S = \{s_1, s_2, \ldots, s_n\}$. As it was mentioned earlier, R is obtained following the procedure of the Kleene theorem proof [96].
Namely, R can be obtained as the union of the RE R^n_{ij}, where s_i is an initial location and s_j is a final one. For any $i, j, k \leq n$, R^k_{ij} is the set of words that lead the automaton A from location s_i to location s_j without passing at locations in $\{s_{k+1} \ldots s_n\}$, and it is defined inductively as: $R^k_{ij} \stackrel{\text{def}}{=} R^{k-1}_{ij} \oplus R^{k-1}_{ik}(R^{k-1}_{kk})^* R^{k-1}_{kj}$; R^0_{ij} is $\{< s_i, a, s_j > :< s_i, a, \psi, \alpha, s_j > \in Edges(A)\}$ if one of such transition exists, it is $\{\epsilon\}$ if $i = j$ and it is the empty set otherwise.[4]

9.5 Verifying Finite TC-RE

In this section, we want to illustrate the formerly introduced concept by showing how to solve the model-checking problem for finite TC-REs. That is, we deal with the case of TC-REs which REs have no occurrence of the star operator (Kleene closed). Finiteness implies that the RE can be rewritten as a finite union of words (RE without Kleene operator or union occurrences). Then $(\sigma, \tau) \models_{\mathcal{P}} \varphi$ where $\varphi \stackrel{\text{def}}{=} \sum_{b \in B} c_b \int b \sim M$ must be checked for all σ in which the RE can be decomposed into (a finite number) and for all $\tau \in Sol(\sigma, \Delta)$. We show how to check a TC-RE whose RE part is a word, and hence we can solve the problem for any finite RE. Note that all words of a finite RE can be checked in parallel.
Given a word σ and a set of time constraints Δ, we can associate to it a set of variables and a set of linear constraints $C(\sigma, \Delta)$ such that the set of solutions is precisely the set $Sol(\sigma, \Delta)$. Formally, the set $C(\sigma, \Delta)$ of inequalities on the variable set $(\tau_i)_{i<|\sigma|}$ is defined by

$$\{\tau_0 = 0\} \cup$$
$$\{\tau_i - \tau_{i-1} \geq 0 \ : \ 0 < i < |\sigma|\} \cup$$
$$\{\tau_i - \tau_{last\_\delta\_in\_\sigma_{i-1}} \sim c \ : \ 0 < i < |\sigma| \wedge (\delta, \sigma_i, \sim c) \in \Delta \wedge \delta \cap \sigma_{i-1}] \neq \emptyset\}$$

[4] Hereafter we will consider that RE do not contain empty set as a proper subexpression (they can be easily eliminated).

Thus checking $\forall \tau \in Sol(\sigma, \Delta) : (\sigma, \tau) \models_{\mathcal{P}} \sum_{b \in B} c_b \int b \sim M$ is exactly the same as checking whether the minimum of the function $f^{\sigma}_{\varphi,\mathcal{P}}(\tau)$ subject to $C(\sigma, \Delta)$ is $\sim$ than M if $\sim \in \{\geq, >\}$, or the maximum of the function $f^{\sigma}_{\varphi,\mathcal{P}}(\tau)$ subject to $C(\sigma, \Delta)$ is $\sim$ than M if $\sim \in \{\leq, <\}$, which can be solved by linear programming techniques.

Example: 20 *Let σ be $(\perp, 0)(0,1)(1,2)(2,1)(1,2)(2,3)$ and let Δ be as in the previous example. Then $C(\sigma, \Delta)$ is the set of the following inequalities:*

$$\begin{aligned}
&\tau_0 = 0,\\
&\tau_1 - \tau_0 \geq 0, \tau_2 - \tau_1 \geq 0, \tau_3 - \tau_2 \geq 0, \tau_4 - \tau_3 \geq 0, \tau_5 - \tau_4 \geq 0,\\
&\tau_5 - \tau_0 \geq 600, \tau_3 - \tau_2 \geq 300.
\end{aligned}$$

Since we know the structure of these systems of constraints, we can outline some procedures which can be applied to simplify the linear programming problem. For example, if one variable stands for a transition where both source and target locations have the same contribution it could be eliminated provided it plays an irrelevant role in the inequality system. That is, the deletion of the transition should lead to a system equivalent to the projection onto the rest of the variables. This can be achieved through efficient existential elimination procedures that take into account just the related variables.
Another improvement of performance could be achieved through a divide and conquer technique. If we find a variable, namely k, such that there is no inequality $\tau_j - \tau_i \sim c \in C(\sigma, \Delta)$ with $i < k < j$ we can divide the linear problem into two independent problems (constraints naming variables indexed less than or equal to k and constraints naming variables indexed greater than or equal to k) and the minimum/maximum is obtained through addition of local minimums/maximums. To simplify the system of constraints, techniques like the ones presented in [114] could be used.

9.6 Infinite TC-RE

In the previous section we show a straightforward way to derive from the timed automaton A a TC-RE that defines the set of timed words of A which lead to a location in F. However, we want a TC-RE with "good" properties which allow a simple treatment of Kleene closed subexpressions in our verification procedure. Such a TC-RE is said to be "well-behaved" TC-RE.
The first step of our model-checking procedure is to obtain a well-behaved TC-RE from the timed automaton to be analyzed. Then, if the obtained TC-RE is finite, we can apply linear programming techniques to solve the problem as shown before. For the infinite case we present, in next sections, some techniques for either inferring that the LDI is violated or reducing the original expression to a finite TC-RE which is equivalent to the original one for the LDI. In the remainder sections we give more details of our idea.

9.6.1 Well-Behaved TC-RE

In this section, we show a special class of TC-RE satisfying properties that simplify our algorithm (for dealing with infinite RE) without sacrificing the expressive power.

Definition 38 *A TC-RE* (R, Δ) *is* **fusion closed** *iff for any sequences over* E_{id} σ, σ' *and* θ, *it holds that* $\sigma\theta \in R$, $\sigma' \in Prefixes(R)$ *and* $tgt(last(\sigma)) = tgt(last(\sigma'))$ *then* $\sigma'\theta \in R$.

This is a natural property of TC-RE resulting from a $\Delta Kleene$ procedure.

Definition 39 *A TC-RE* (R, Δ) *is* **feasible** *iff* $\forall \sigma \in R : Sol(\sigma, \Delta) \neq \emptyset$.

This property means that any word described by the RE can be "time stamped" in such a way that the constraints imposed by Δ are satisfied.

Definition 40 *A TC-RE (R, Δ) is* **no-deadline constrained iterations** *iff* $\forall \sigma \in R, \forall k < |\sigma|, \forall (\delta, \sigma_k, \leq c) \in \Delta : \delta \cap \sigma_{k-1]} \neq \emptyset, \neg [\exists i, j : last\_\delta\_in\_\sigma_{k-1]} < i < j < k : Repeatable(\sigma_{[i,j]})]$.

Definition 41 *A TC-RE (R, Δ) is* **no-delay constrained iterations** *iff* $\forall \sigma \in R, \forall k < |\sigma|, \forall (\delta, \sigma_k, \geq c) \in \Delta : \delta \cap \sigma_{k-1]} \neq \emptyset, \forall i, j : last\_\delta\_in\_\sigma_{k-1]} < i < j < k \wedge Repeatable(\sigma_{[i,j]}) : MIN_\delta(\sigma_{i-1]}) > c$.

The last two properties have a more technical nature and simplify the analysis of star subexpressions (Kleene closed). Intuitively, the property **no-deadline constrained iterations** rules deadlines ranging over a repeatable sub-word.
On the other hand, the property **no-delay constrained iterations** that delay tests for clocks not reset in a previous repeatable subword are redundant. In fact, the minimum values for those clocks are greater than those test constants even before the repeatable subword.
Roughly speaking, these properties imply that the number of iterations does not affect, and is not affected by timing constraints. Thus our algorithm can, in some extent, analyze Kleene closed subexpressions locally.

Definition 42 *A TC-RE is* **well-behaved** *iff it is fusion closed, feasible, no-deadline constrained iterations and no-delay constrained iterations.*

Example: 21 *Let*

$$
\begin{aligned}
R &= (\perp, 0)(0,1)\,((1,2)(2,1))^*\,(1,3) \\
\Delta &= \left\{ \begin{array}{lll} (\{(\perp,0)\}, & (0,1), & \geq 100), \\ (\{(0,1),(2,1)\}, & (1,2), & \geq 50), \\ (\{(\perp,0)\}, & (1,3), & \geq 100) \end{array} \right\}
\end{aligned}
$$

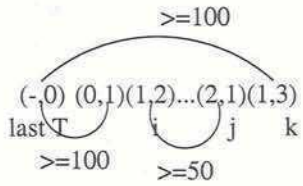


Figure 9.2: No-Delay Constrained Iterations

Then (R, Δ) *is a well-behaved TC-RE. It is fusion closed, feasible and transition* $(1,3)$ *tests for delay that is already true in (0,1) (see figure 9.2).*

Remember we are trying to verify if $\Delta Kleene(A, F) \models_{\mathcal{P}} \varphi$. But, is the obtained TC-RE well-behaved ? Since transitions go from and to locations, the resulting language has the fusion closed property.
But the last three properties of well-behavior are not guaranteed if we apply the Kleene procedure to an arbitrary timed automaton (the last two conditions are not true, in particular, when there are timing constraints covering cycles). For instance, in the composition of both automata of the working case has infeasible paths and, moreover clock z constraints the number of iterations of cycles.

Fortunately, a property called **reachability equivalence** will ensure the well-behaveness of a TC-RE. Some of the symbolic state space representations (e.g. [104]) produce automata where all paths are feasible and, moreover, for which "reachability equivalence" holds. Given a timed automaton, these techniques produce an automaton which represents all the reachable timed states of the original one. Generally, the resulting automaton is much smaller than the region graph. The reachability graph [104] is among such kind of automata.

To understand why the reachability graph solves our problem of getting a well-behaved TC-RE, lets first define the reachability equivalence property.

The reachability equivalence property means that the minimum (resp. maximum) value for a clock δ that may be tested in the future is the same for all paths leading to a location l. This minimum (resp. maximum) will be denoted MIN_{δ}^{l} (resp. MAX_{δ}^{l}) [5]. Note that all values greater than the maximum constant appearing in a time constraint in Δ are considered the same, and could be noted ∞. Then we can check only one path leading to l.

Definition 43 *Let (R, Δ) be a TC-RE. A clock δ has a* **test after location** *l, denoted* $FutureTest_{\delta}(l)$, *iff* $\exists \sigma \in R, \exists j,k \in \mathrm{N}$ *such that* $last\_\delta\_in\_\sigma_{k-1]} \leq j < k < |\sigma| \ \wedge \ l = tgt(\sigma_j) \ \wedge \ (\delta, \sigma_k, \sim c) \in \Delta$.

[5] Actually, this is a simplified version of the reachability equivalence property. In [29] we also require the same property for the difference of each pair of clocks which might be tested in the future.

Definition 44 *A TC-RE (R, Δ) satisfies the* **reachability equivalence** *property iff for all clocks δ, for all locations l such that $FutureTest_\delta(l)$, for all $\theta, \theta' \in Prefixes(R)$ such that $tgt(last(\theta)) = tgt(last(\theta')) = l$, then*

- $MIN_\delta(\theta) = MIN_\delta(\theta') \wedge$
- $MAX_\delta(\theta) = MAX_\delta(\theta')$

Example: 22 *Suppose we have two histories leading to location 2 of the environment automaton: $(\perp, 0)(0, 1)(1, 2)$ and $(\perp, 0)(0, 1)(1, 2)(2, 1)(1, 2)$. They have the same values for that minimum (maximum) of clock x. On the other hand, $(\perp, 0)(0, 1)$ and $(\perp, 0)(0, 1)(1, 2)(2, 1)$ are not considered equivalent since they do not share the same value (in the first history the minimum value is 0 while in the second one the minimum is 300).*

The next lemmas answer our question.

Lemma 11 *Reachability equivalence and strong non-zenoness imply no-deadline constrained iterations.*

Proof 18 *See Sect. 9.11.*

Lemma 12 *Reachability equivalence, strong non-zenoness and fusion closed imply no-delay constrained iterations.*

Proof 19 *See Sect. 9.11.*

In the next section, we show how to obtain a well-behaved TC-RE from a timed automaton using the reachability graph. This TC-RE will recognize the relevant language of timed words leading to a final location (the timed words to be analyzed for our validity problem).

9.7 Problem Transformation in terms of Well-Behaved TC-RE

Let us illustrate the idea of using the reachability graph technique. The reachability graph is a well known concept in many timed formalisms [99, 25]. In particular, given a timed automaton A, the reachability graph $RG(A)$ of A can be seen as an automaton that accepts (up to renaming, let us call it $\beta : Locs(RG(A)) \rightarrow Locs(A)$) the same language as A. It is built by unfolding symbolically the original graph in such a way that the language accepted by the underlying graph is feasible. This implies immediately the **feasibility** of its associated TC-RE.

Moreover, the procedure equates paths when they satisfy the **reachability equivalence** property [104]. Let us sketch the conceptual phases of the RG construction (for more details see [104]). The first stage is the definition of an infinite tree-shaped automaton whose locations are the feasible sequences of transitions of the original automaton (elsewhere called histories). Then, a sufficient condition is used to identify strong transition bisimilar histories. To define such condition, they remarked that given a history it is easy to calculate for each clock the minimum and the maximum values on the set of runs exercising that sequence of transitions (remember $Times_\delta(\theta)$).[6] Then the condition says that two leading to the same location are equivalent if also those calculated minimum (maximum) for each clock are the same in both paths or alternatively they are greater than any constant that could occur in a future comparison.

Given $\beta : Locs(RG(A)) \rightarrow Locs(A)$, we define

- If $F \subseteq Locs(A)$ are the final locations of A, the set of final locations of $RG(A)$, denoted $\beta^{-1}(F)$, is the set of locations $l \in Locs(RG(A))$ such that $\beta(l) \in F$,
- If $\mathcal{P} : Props \rightarrow 2^{Locs(A)}$ is the mapping for A, the mapping $\beta^{-1}(\mathcal{P}) : Props \rightarrow 2^{Locs(RG(A))}$ for $RG(A)$ is $\beta^{-1}(\mathcal{P})(pr) = \beta^{-1}(\mathcal{P}(pr))$ for $pr \in Props$.

Fact 8 *There is a mapping $\beta : Locs(RG(A)) \rightarrow Locs(A)$ such that:*

1. $\forall(\sigma,\tau) \in L(\Delta Kleene(RG(A),\beta^{-1}(F))) : (\beta_*(\sigma),\tau) \in L(\Delta Kleene(A,F))$
2. $\forall(\sigma,\tau) \in L(\Delta Kleene(A,F)), \exists\sigma' : \beta_*(\sigma') = \sigma \wedge (\sigma',\tau) \in L(\Delta Kleene(RG(A),\beta^{-1}(F)))$

where β_* is the natural extension induced by β to sequences.

Lemma 13 *$\Delta Kleene(RG(A),\beta^{-1}(F))$ satisfies the reachability equivalence property.*

Now we formulate the main result of this section.

Theorem 11 *Given a timed automaton A, a set of final locations $F \subseteq Locs(A)$, a valuation $\mathcal{P} : Props \rightarrow 2^{Locs(A)}$ and a LDI formula φ then*

1. *$\Delta Kleene(RG(A),\beta^{-1}(F))$ is a well-behaved TC-RE,*
2. *$\Delta Kleene(RG(A),\beta^{-1}(F)) \models_{\beta^{-1}(\mathcal{P})} \varphi$ iff $(A,F) \models_{\mathcal{P}} \varphi$.*

Proof 20 *It follows immediately from the Theorem 10, Fact 8 and Lemmas 11 and 12.*

[6] These values are calculated trough a longest path algorithm on a weighted graph which nodes are the transitions and the edges comes from the linear inequalities [104, 99]

Then the well-behaved TC-RE we must built is $\Delta Kleene(RG(A), \beta^{-1}(F))$. Therefore in the rest of the chapter we can deal only with the problem of deciding whether $(R, \Delta) \models_{\mathcal{P}} \varphi$ for a well-behaved TC-RE (R, Δ).

Definition 45 *A word over E_{id} , σ is* **non-transient** *for Δ, denoted $NonTransient_{\Delta}(\sigma)$, iff $\exists k < |\sigma| : \exists(\delta, \sigma_k, \geq c) \in \Delta : \delta \cap \sigma \neq \emptyset \wedge c > 0$.*

This means that if a non-transient repeatable word is infinitely iterated then time must diverge.

Definition 46 *A well-behaved TC-RE (R, Δ) is* **strongly non-zeno** *iff $\forall \sigma \in R : \forall\ 0 \leq i \leq j < |\sigma| : Repeatable(\sigma_{[i,j]}) \Rightarrow NonTransient_{\Delta}(\sigma_{[i,j]})$.*

Theorem 12 *If a timed automaton A is strongly non-zeno then $\Delta Kleene(RG(A), \beta^{-1}(F))$ is strongly non-zeno.*

Proof 21 *Supoose that we have a transient repeatable subword of $\Delta Kleene(RG(A), \beta^{-1}(F))$ (which is, indeed, a cycle of $RG(A)$ and thus a cycle of A up to renaming).*

We can build an infinite run of A as follows: take τ of $Sol(\sigma_j \sigma_{[i,j]\sigma_{[j}}, \Delta)$ (due to feasibility there exists such a temporization). It is easy to see that $\tau_{j]} \lhd (0...0) \in Sol(\sigma_j \sigma_{[i,j]}, \Delta)$. Infact, if a clock is tested for delay and was reset outside cycle, that test is already true in the first iteration (which is included in $\sigma_{j]}$). If the deadlines were true in τ they must be also true in $\tau_{j]} \lhd (0...0)$ (less time elapses), and due to transientness of $\sigma_{[i,j]}$ there is no delay test using clocks reset in that word. This reasoning serves to conclude that we can build a non divergent run of A by repeating infinitely often $\sigma_{[i,j]}$ with no time elapse. This contradicts strongly non-zenoness of A.

Example: 23 *The reachability graph for the SUA automaton is shown in Figure 9.3. The pairs indicate the value of the β renaming.*

A TC-RE that accepts all the time words of the RG in Figure 9.3, which lead to the final location 4 is the following:

$$
\begin{array}{lcl}
R & \stackrel{def}{=} & (0,1)(1,2)R^{10}_{2,4} \\
R^{10}_{2,4} & \stackrel{def}{=} & (2,4) \oplus (2,3)R^{10}_{3,4} \\
R^{10}_{3,4} & \stackrel{def}{=} & (3,4) \oplus (3,5)(5,7)R^{10}_{7,4} \\
R^{10}_{7,4} & \stackrel{def}{=} & (7,4) \oplus (7,8)R^{10}_{8,4} \\
R^{10}_{8,4} & \stackrel{def}{=} & (8,4) \oplus (8,9)(9,10){R^{10}_{10,10}}^{*}[(10,4) \oplus (10,8)(8,4)] \\
R^{10}_{10,10} & \stackrel{def}{=} & (10,8)(8,9)(9,10)
\end{array}
$$

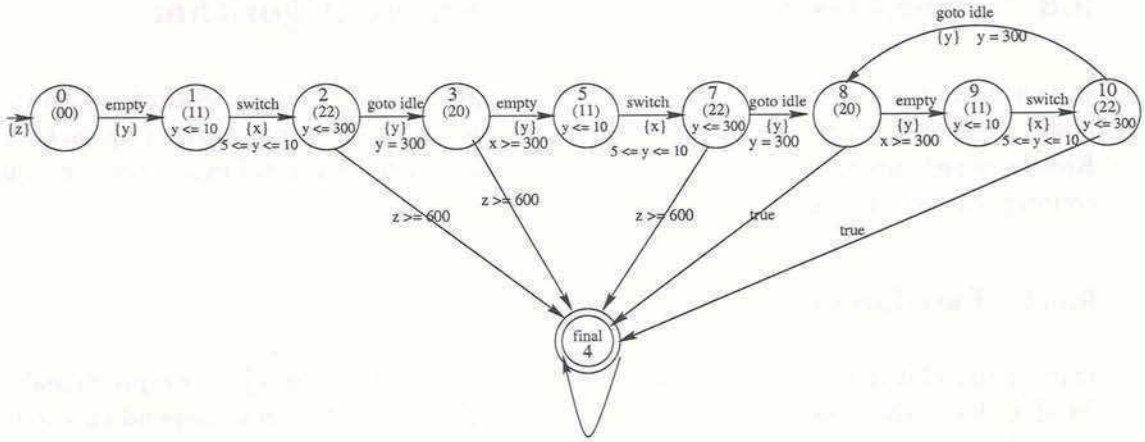


Figure 9.3: The Reachability Graph

and

$$\Delta \stackrel{def}{=} \left\{ \begin{array}{lll} (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (1,2), & \leq 10), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (1,2), & \geq 5), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (2,4), & \leq 300), \\ (\{(\bot,0)\}, & (2,4), & \geq 600), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (2,3), & = 300), \\ (\{(\bot,0)\}, & (3,4), & \geq 600), \\ (\{(1,2),(5,7),(9,10)\}, & (3,5), & \geq 300), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (5,7), & \leq 10), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (5,7), & \geq 5), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (7,4), & \leq 300), \\ (\{(\bot,0)\}, & (7,4), & \geq 600), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (7,8), & = 300), \\ (\{(1,2),(5,7),(9,10)\}, & (8,9), & \geq 300), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (9,10), & \leq 10), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (9,10), & \geq 5), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (10,4), & \leq 300), \\ (\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}, & (10,8), & = 300) \end{array} \right\}$$

In principle, ΔKleene does not produce economical TC-RE in the number of resets. Indeed, in this examples many resets are unnecessary in the tuples due to graph topology. For example, $(0,1)$ is never the last reset of the set $\{(0,1),(2,3),(3,5),(7,8),(8,9),(10,8)\}$ for transition $(10,8)$.

9.8 Principles for the model-checking Algorithm

In this section we present some necessary concepts to explain the checking algorithm. As it will be shown later in this chapter, our algorithm processes, in a bottom up fashion, the Kleene closed subexpressions trying to reduce them into a finite subexpression or to find a counterexample, i.e. a violation of the LDI.

9.8.1 Past-Independence

Due to the global nature of timing constraints, it is hard to analyze compositionally the TC-RE. Valid time assignments for a word having a past context may depend on the values of clocks which are tested with no previous reset in the word. These clocks are called **free clocks**.

Definition 47 (Free Clocks) $FreeClocks(\sigma, \Delta) \stackrel{def}{=} \{\delta \mid 0 \leq i < |\sigma| \wedge (\delta, \sigma_i, \sim c) \in \Delta \wedge (i = 0 \vee \delta \cap \sigma_{i-1]} = \emptyset)\}$.

However, there are some cases where a word can be analyzed locally and then some valid conclusions can be drawn.

Definition 48 *Let (R, Δ) be a TC-RE. We say that θ is* **past-independent** *iff $MIN_{\delta}^{src(\theta_0)} = MAX_{\delta}^{src(\theta_0)}$ for all $\delta \in FreeClocks(\theta, \Delta)$.*

The past-independence means that the minimum and maximum values for the free clocks of θ coincide then the values are known constants when entering the first location of the word (i.e., the free clocks have fixed values and the word becomes "context free").

Definition 49 *Let (R, Δ) be a TC-RE. A* **context** *for θ is a pair (θ', τ') such that $\theta'\theta \in Prefixes(R) \wedge \tau' \in Sol(\theta', \Delta)$.*

Corollary 8 *Let (R, Δ) be a TC-RE. If θ is past-independent then for all time sequences τ it holds that $\exists$ context (θ', τ') for $\theta : \tau' \lhd \tau \in Sol(\theta'\theta, \Delta)$ implies that $\forall$ context $(\theta", \tau")$ for $\theta : \tau" \lhd \tau \in Sol(\theta"\theta, \Delta)$.*

The past-independence implies that a solution τ for a valid past context (θ', τ') for a word θ is a solution for any valid past context $(\theta", \tau")$ for the word θ.

If a word θ is past-independent then the following inequality system $\hat{C}(\theta, \Delta)$ on the variable set $(\gamma_i)_{i<|\theta|}$ provides all the valid relative time assignments γ for θ as subword of a word

in R

$$\begin{array}{l}\{\gamma_i - \gamma_{i-1} \geq 0 \ : \ 0 < i < |\theta|\} \cup \\ \left\{\gamma_i - \gamma_{last\_\delta\_in\_\theta_{i-1}} \sim c \ : \ 0 < i < |\theta| \wedge (\delta, \theta_i, \sim c) \in \Delta \wedge \delta \cap \theta_{i-1}] \neq \emptyset \right\} \cup \\ \left\{MIN_{\delta}^{src(\theta_0)} + \gamma_i \sim c \ : \ 0 \leq i < |\theta| \wedge (\delta, \theta_i, \sim c) \in \Delta \wedge (i = o \vee \delta \cap \theta_{i-1}] = \emptyset) \right\}\end{array}$$

The corollary 8 can be extended:

Corollary 9 *If θ is past-independent then it holds that $\forall \tau$ solution of $\hat{C}(\theta, \Delta)$: $\forall$ context (θ', τ') for $\theta : \tau' \lhd \tau \in Sol(\theta'\theta, \Delta)$.*

The following results are the basis for the correctness of the algorithm shown in the next section. Firstly, we show the results when $\sim \in \{\leq, <\}$. For $\sim \in \{\geq, >\}$ are analogous.

Theorem 13 *Let (R, Δ) be a strongly non-zeno and well-behaved TC-RE, let A^* be a subexpression of R where $\theta \in A$ is a past-independent word. If the maximum value for the $f^{\theta}_{\varphi,\mathcal{P}}(\gamma)$ subject to $\hat{C}(\theta, \Delta)$ is greater than zero and the test $\sim \in \{\leq, <\}$ then $(R, \Delta) \not\models_{\mathcal{P}} \varphi$.*

Proof 22 *See Sect. 9.11.*

That is, if we detect a past-independent word θ of a Kleene closed subexpression such that the maximum value for the duration expression is greater than zero, we can conclude that the LDI is violated by the whole expression. In fact, the duration expression will not be bounded (because that timed word can be repeated unboundedly).

Theorem 14 *Let (R, Δ) be a strongly non-zeno and well-behaved TC-RE, let A^* be a subexpression of R such that all $\theta \in A$ are past-independent words. If the maximum value for the duration expression $f^{\theta}_{\varphi,\mathcal{P}}(\gamma)$ subject to $\hat{C}(\theta, \Delta)$ is less than or equal to zero and the test $\sim \in \{\leq, <\}$ then, by replacing A^* in R with $\epsilon \oplus A$, we get a regular expression R' for which $(R, \Delta) \models_{\mathcal{P}} \varphi$ iff $(R', \Delta) \models_{\mathcal{P}} \varphi$.*

Proof 23 *See Sect. 9.11.*

This complementary result states that the Kleene closed subexpression can be replaced by some unfolds of the subexpression in order to obtain an equivalent expression w.r.t. the LDI (the repetitions do not contribute to the duration expression),. Analogously,

Theorem 15 *Let (R, Δ) be a strongly non-zeno and well-behaved TC-RE, let A^* be a subexpression of R where $\theta \in A$ is a past-independent word. If the minimum value for the $f^{\theta}_{\varphi,\mathcal{P}}(\gamma)$ subject to $\hat{C}(\theta, \Delta)$ is lower than zero and the test $\sim \in \{\geq, >\}$ then $(R, \Delta) \not\models_{\mathcal{P}} \varphi$.*

Proof 24 *Analogous to theorem 13.*

Theorem 16 *Let (R, Δ) be a strongly non-zeno and well-behaved TC-RE, let A^* be a subexpression of R such that all $\theta \in A$ are past-independent words. If the minimum value for the duration expression $f^{\theta}_{\varphi,\mathcal{P}}(\gamma)$ subject to $\hat{C}(\theta, \Delta)$ is greater than or equal to zero and the test $\sim \in \{\geq, >\}$ then, by replacing A^* in R with $\epsilon \oplus A$, we get a regular expression R' for which $(R, \Delta) \models_{\mathcal{P}} \varphi$ iff $(R', \Delta) \models_{\mathcal{P}} \varphi$.*

Proof 25 *Analogous to theorem 14.*

Summarizing these four theorems

Check	$\sim \in$	$(R, \Delta) \models_{\mathcal{P}} \varphi$
maximum $(f^{\theta}_{\varphi,\mathcal{P}}(\gamma)) > 0$	$\{\leq, <\}$	false
maximum $(f^{\theta}_{\varphi,\mathcal{P}}(\gamma)) \leq 0$	$\{\leq, <\}$	replace R by R'
minimum $(f^{\theta}_{\varphi,\mathcal{P}}(\gamma)) < 0$	$\{\geq, >\}$	false
minimum $(f^{\theta}_{\varphi,\mathcal{P}}(\gamma)) \geq 0$	$\{\geq, >\}$	replace R by R'

9.8.2 Obtaining Past-Independent Iterations

This section proposes some manipulations that can be used to obtain past-independent iterations from dependent ones:

Rotation. To obtain past-independent iterations, it may be applied the rewriting rule $(AB)^* \equiv A(BA)^*B \oplus \epsilon$ (a rotation). Obviously, that manipulation does not guarantee past-independence.

Constraint Elimination. If rotation fails, there is a conservative treatment based on the elimination of non-fixed free clocks from the iterations. The basic idea is to eliminate non fixed free clocks from the iterations to make them past-independent. Let σ_i be a transition such that there is a $(\delta, \sigma_i, \sim c) \in \Delta$ for which $\delta \cap \sigma_{i-1]} = \emptyset$ and $MIN_{\delta}^{src(\sigma_0)} \neq MAX_{\delta}^{src(\sigma_0)}$. Then we can simply eliminate these kind of tuples from Δ to achieve the past-independency [7].

Tuple elimination produces a TC-RE which language is a superset of the original one (see Fact 4). Therefore, this step is conservative. If the duration is valid over a larger language it is also valid over the original one. On the other hand, any counterexample found using a so altered subexpression must be checked for membership. Thus, in some cases the method is not able to return a yes/no response (unconclusive: "don't know!").

[7] Actually, the procedure first renames the transition σ_i producing a new (R', Δ') and a new mapping $\mathcal{P}'$ such that: $(R', \Delta') \models_{\mathcal{P}'} \varphi$ iff $(R, \Delta) \models_{\mathcal{P}} \varphi$ (see [29]). This renaming is done to achieve a local elimination. Otherwise, tests which are not troublesome for past-independence would be eliminated.

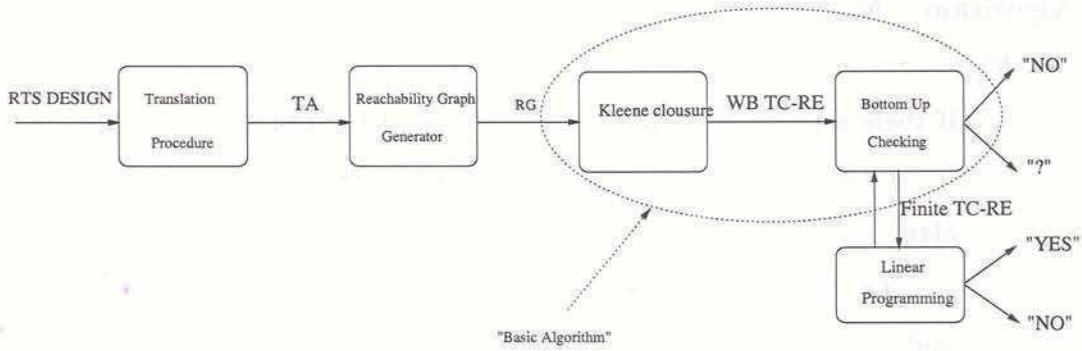


Figure 9.4: The Conceptual Architecture

9.9 The Basic Algorithm

We have shown in previous sections how the original problem $(A, F) \models_{\mathcal{P}} \varphi$ boils down into a verification in a well-behaved TC-RE $(R, \Delta) \models_{\mathcal{P}} \varphi$. Given the well-behaved TC-RE, a theoretical algorithm would process, in a bottom up fashion, the obtained Kleene closed subexpressions. Under the hypothesis of Theorems 13, 14, 15 and 16 it is possible to either reduce the TC-RE into a finite expression or find a violation of the LDI. A scheme for our whole method is shown in Figure 9.4.

In this section, we present a sketch of an on-the-fly algorithm that shows that it is not necessary to translate the whole automaton $RG(A)$ to the TC-RE at the beginning. We use an n^3-matrix to store $R^k_{i,j}$. The input is a strongly-non zeno TA such that its iterations can be written into past independent ones. [8]

[8] Remember that there are conservative treatments to convert them into past independent subwords as explained in the last section. Elimination of constraints can be included in the presented algorithm as well.

Algorithm: **for** $i, j \leq n$

$R_{ij}^0 :=$

if there is an edge $< s_i, a, \psi, \alpha, s_j > \in Edges(A)$ **then**

$\{< s_i, a, s_j > : < s_i, a, \psi, \alpha, s_j > \in Edges(A)\}$

else

if $i = j$ **then** $\{\epsilon\}$ **else** $\emptyset$ **end-if**

end-if

end-for

for $k = 1$ *to* n

if $R_{kk}^{k-1} = \emptyset$ **then**

$Cicle_k := \emptyset$

else

let A, B such that $AB = R_{kk}^{k-1}$

if $\exists \theta \in BA$ that satisfy the conditions of Theorems 13 or 15 **then**

exit with result **counterexample was found**

else

if $(A = \epsilon)$ or $(B = \epsilon)$ **then** (no rotation)

$Cycle_k := \epsilon \oplus R_{kk}^{k-1}$

else

$Cycle_k := \epsilon \oplus AB \oplus ABAB$ [9]

end-if

end-if

end-if

for $i, j \leq n$

$R_{ij}^k := R_{ij}^{k-1} \oplus R_{ik}^{k-1} \; Cicle_k \; R_{kj}^{k-1}$

end-for

end-for

[9] $(AB)^* = \epsilon \oplus A(BA)^*B$, and then applying theorems 14 or 16 $(AB)^* = \epsilon \oplus A(\epsilon \oplus BA)B = \epsilon \oplus AB \oplus ABAB$.

Finally the regular expression is reduced [10].

Hence, from Theorems 13, 14, 15 and 16, with this procedure

- we either obtain a finite TC-RE (no occurrence of Kleene closed) that is equivalent up to the duration problem to the original one (see its treatment in Sect. 9.5),
- or we discover a counterexample [11].

Note that the calculus of each $R^k_{i,j}$ for a fixed k can be done in parallel. It is also possible to chose among several strategies of calculus reuse for the sets $R^k_{i,j}$ (e.g. either the sets of words can be stored explicitly in the matrix in order to avoid recalculation or they can be stored symbolically and calculated by demand).

Example: 24 *Consider the automaton in Figure 9.3. Observe that the algorithm analyzes* $R^{10}_{10,10} = (10,8)(8,9)(9,10)$ *(a Kleene closed subexpression). The cycle* $R^{10}_{10,10}$ *is not past-independent. Moreover, if its free clocks,* x *and* y*, are eliminated, the maximum of the duration expression is greater than 0. Thus, that consevative manipulation would be inconclusive and a rotation should be tried first.*
Indeed, $R^{10}_{10,10}{}^{*} = \epsilon \oplus [(10,8)R^{10}_{8,8}{}^{*}(8,9)(9,10)]$ *where* $R^{10}_{8,8} = (8,9)(9,10)(10,8)$*, is a rotation. Again* $R^{10}_{8,8}$ *is past-dependent. However, if the constrains on the free clock* x *are eliminated the following system is obtained:*

$$\begin{aligned} &5 \leq \tau_1 - \tau_0 \leq 10, \\ &\tau_2 - \tau_0 = 300 \end{aligned}$$

And, fortunately, the maximum value of the objective function for the LDI (in Example 4) subject to that system of constraints is $19 * 10 - 1 * 290 = -100$*. Then, due to theorem 14, the algorithm replaces* $R^{10}_{8,8}{}^{*}$ *with* $\epsilon \oplus R^{10}_{8,8}$ *obtaining the following sets of finite words:*

$$\left\{ \begin{array}{l} (0,1)(1,2)(2,4), \\ (0,1)(1,2)(2,3)(3,4), \\ (0,1)(1,2)(2,3)(3,5)(5,7)(7,4), \\ (0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,4), \\ (0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,4), \\ (0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,4), \\ (0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,4), \\ (0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,8)(8,4), \\ (0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,4), \\ (0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,8)(8,4) \end{array} \right\}$$

[10] Using rules like $\epsilon A = A\epsilon = A, (\epsilon \oplus A)^* = (A \oplus \epsilon)^* = A^*, A \oplus A = A$.

[11] Remember that, if a conservative treatment was applied to make iterations past independent counterexamples must be tested for membership. If membership does not hold the procedure is inconclusive.

Calculating the maximum objective function on these words the algorithm concludes that the duration property is satisfied. In fact, lets make some observations to conclude the same result. Let $\tau_{i,j}$ the variables that stand for the time of occurrence of (i,j). Firstly, note that in all cases the positive contributions (19) are in the following intervals $[\tau_{1,2}, \tau_{0,1}]$,$[\tau_{5,7}, \tau_{3,5}]$, and $[\tau_{9,10}, \tau_{8,9}]$. The maximum value that each interval can contribute is 190 *(they length is at most 10). Therefore, since time length of words is at least 600 any of the former words showing two of those intervals have a maximum less than or equal to (-1)*500 + 2* 190 = - 120. That leaves to the analysis the following words:*

$$\left\{\begin{array}{l}(0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,4),\\(0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,4),\\(0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,4),\\(0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,8)(8,4),\\(0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,4),\\(0,1)(1,2)(2,3)(3,5)(5,7)(7,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,8)(8,9)(9,10)(10,8)(8,4)\end{array}\right\}$$

Secondly, If we have variable $\tau_{2,3}$ involved in the system, since $\tau_{2,3} - \tau_{0,1} = 300$ the contribution of location 2 *is always less than or equal to -290. The same is true for $\tau_{7,8}$ with location* 7*, and $\tau_{10,8}$ with location* 10*. For each word, we can count the number of positive locations (*1,5 *and* 9*) which contributes at most* 190 *each against the occurrences of $\tau_{2,3}, \tau_{7,8}$ and $\tau_{10,8}$ that contributes less than or equal to -290. We can conclude for all words the maximum is less than zero.*

9.10 Conclusions and Discussions

We have presented a conservative procedure that analyzes some paths of the Reachability Graph [104] (generally much smaller than the region graph) using linear programming. The procedure works for strongly non-zeno timed automaton A. Let us remark some facts about the technique:

- it is accurate when the TC-RE is finite. That happens, for example, when there are deadlines to arrive to final locations,
- its is also accurate when cycles can be rewritten in terms of past-independent iterations. Some identified classes such as the Alternating RQ automata fulfill the condition of independent iterations [113]. In general, this condition is also satisfied when the tests in a cycle are always preceded by a reset in the iteration,
- in general it can verify conservatively the whole class of strongly non-zeno timed automata avoiding digitization (i.e., building the region graph), and
- we believe that in the near future we will be able to verify accurately the whole class of strongly non-zeno timed automata adding a local digitization technique to convert

non past-independent iterations into independent ones (using the possible integer values of free clocks).

The practicality of the algorithm is still an empirical exercise. Although the estimated worst case complexity is not better than the procedure shown in [106] there are some encouraging observations. First of all, techniques like [104] produce graphs which are, generally, much smaller than the Region Graph. The complexity of our RE conversion procedure seems to depend on the number of transitions and it is not necessarily exponential on the number of nodes. Also, the algorithm works on the fly and many steps can be done in parallel. Finally, we use linear programming instead of integer-linear programming. We were able to check small but yet interesting examples.

Note also that, since the work is based on the reachability graph construction, it could be easily migrated to other formalisms like real-time versions of Petri Nets [25] and ModeCharts [99].

9.11 Proofs of Lemmas and Theorems

Lemma 11 Reachability equivalence and strong non-zenoness imply no-deadline constrained iterations.

Proof 26 *Suppose that the property no-deadline constrained iterations is violated. That means* $\exists \sigma \in R$, $\exists k < |\sigma|$, $\exists (\delta, \sigma_k, \leq c) \in \Delta : \delta \cap \sigma_{k-1]} \neq \emptyset$, $\exists i, j : last\_\delta\_in\_\sigma_{k-1]} < i < j < k \ \wedge \ Repeatable(\sigma_{[i,j]})$.

$last\_\delta\_in\_\sigma_{k-1]} < i$ *means that reset of* δ *occurs before the step* i. *Then there is no reset of* δ *in* $\sigma_{[i,k]}$.

Since $last\_\delta\_in\_\sigma_{k-1]} < j < k$ *and* $(\delta, \sigma_k, \leq c) \in \Delta$ *then it holds* $FutureTest_\delta(tgt(\sigma_j))$. *But* $\sigma_{i-1]}$ *and* $\sigma_{j]}$ *are sequences ending on the same location* $tgt(\sigma_j)$, *because of* $Repeatable(\sigma_{[i,j]})$. *By reachability equivalence,* $MIN_\delta(\sigma_{i-1]}) = MIN_\delta(\sigma_{j]})$.

Due to feasibility, $\exists \tau \in Sol(\sigma, \Delta)$. *Since* $(\sigma, \tau) \in L(R, \Delta)$, δ *is no reset in* $\sigma_{[i,k]}$ *and* σ *has a test* $\leq c$ *for the clock* δ *in location* $tgt(\sigma_k)$ *then both* $MIN_\delta(\sigma_{i-1]})$ *and* $MIN_\delta(\sigma_{j]})$ *are not* ∞. *So* $MinTimes_\delta(\sigma_{i-1]}) = MinTimes_\delta(\sigma_{j]})$ *(1).*

But loop in $\sigma_{[i,j]}$ *is non-transient because of strongly non-zenoness, and the clock* δ *is not reset in* $\sigma_{[i,j]}$. *Then by Lemma 14 we conclude that* $MinTimes_\delta(\sigma_{i-1]}) < MinTimes_\delta(\sigma_{j]})$, *opposite to (1).*

Lemma 12 Reachability equivalence, strong non-zenoness and fusion closed imply no-delay constrained iterations.

Proof 27 *Let* $\sigma \in R$, $k < |\sigma|$, $(\delta, \sigma_k, \geq c) \in \Delta$, $\delta \cap \sigma_{k-1]} \neq \emptyset$, *and* $last\_\delta\_in\_\sigma_{k-1]} < i < j < k \ \wedge \ Repeatable(\sigma_{[i,j]})$. *We have to prove that* $MIN_\delta(\sigma_{i-1]}) > c$.

last_δ_in_σ$_{k-1]}$ < i means that reset of δ occurs before the step i. Then there is no reset of δ in σ$_{[i,k]}$. Since there is no reset of δ in σ$_{[i,j]}$, we get by Lemma 14 $MinTimes_\delta(\sigma_{i-1]}) \leq MinTimes_\delta(\sigma_{j]})$ (1).

$Repeatable(\sigma_{[i,j]}) \Rightarrow src(\sigma_i) = tgt(\sigma_j)$, then the concatenation $\sigma_{j]}\sigma_{[i,j]}$ is defined. Also, by fusion clousure, $\sigma_{j]}\sigma_{[i,j]} \in Prefixes(R)$. By Lemma 14, because $\sigma_{[i,j]}$ is non-transient, we get $MinTimes_\delta(\sigma_{j]}) < MinTimes_\delta(\sigma_{j]}\sigma_{[i,j]})$ (2).

From (1) and (2), we know that $MinTimes_\delta(\sigma_{i-1]}) < MinTimes_\delta(\sigma_{j]}\sigma_{[i,j]})$ (3).

last_δ_in_σ$_{k-1]}$ < j < k and $(\delta, \sigma_k, \geq c) \in \Delta$ then it holds $FutureTest_\delta(tgt(\sigma_j))$. But $\sigma_{i-1]}$ and $\sigma_{j]}\sigma_{[i,j]}$ are sequences ending on the same location $tgt(\sigma_j)$. By reachability equivalence, $MIN_\delta(\sigma_{i-1]}) = MIN_\delta(\sigma_{j]}\sigma_{[i,j]})$. Because of (3) it is only posible iff both are equal to ∞. Therefore, $MIN_\delta(\sigma_{i-1]}) > c$.

Lemma 14 *∀ clock δ, ∀θ, θ' such that $\theta\theta' \in Prefixes(R)$ and θ' doesn't reset clock δ:*

1. $MinTimes_\delta(\theta) \leq MinTimes_\delta(\theta\theta')$
2. *If θ' is non-transient then $MinTimes_\delta(\theta) < MinTimes_\delta(\theta\theta')$*

Proof 28 *Let $y \in Times_\delta(\theta\theta')$. Then $\exists x \in Times_\delta(\theta)$ such that $x \leq y$, because θ' does not reset δ. But $MinTimes_\delta(\theta) \leq x$ because $x \in Times_\delta(\theta)$. Then $MinTimes_\delta(\theta) \leq y$ ∀y. Therefore $MinTimes_\delta(\theta) \leq MinTimes_\delta(\theta\theta')$.*

And if θ' is non-transient then it elapse time. So we conclude that $MinTimes_\delta(\theta) < MinTimes_\delta(\theta\theta')$.

Theorem 13 Let (R, Δ) be a strongly non-zeno and well-behaved TC-RE, let A^* be a subexpression of R where $\theta \in A$ is a past-independent word. If the maximum value for the $f^{\theta}_{\varphi,\mathcal{P}}(\gamma)$ subject to $\hat{C}(\theta, \Delta)$ is greater than zero and the test $\sim \in \{\leq, <\}$ then $(R, \Delta) \not\models_{\mathcal{P}} \varphi$.

Proof 29 *We will construct a timed-word such that it will belong in $L(R, \Delta)$ but it not satisfies φ.*

We know by hypothesis that $\exists \sigma, \sigma' : \sigma\theta\sigma' \in R$. Also because of feasibility $\exists \tau = \tau_\sigma \lhd \tau_\theta \lhd \tau_{\sigma'} \in Sol(\sigma\theta\sigma', \Delta)$ with $|\tau_\sigma| = |\sigma|, |\tau_\theta| = |\theta| \wedge |\tau_{\sigma'}| = |\sigma'|$. The idea is to insert the loop θ in this word many times as necesary so that it not satisfies φ.

Because of the fusion closed property, $\sigma\theta^{n+1}\sigma' = \sigma\theta^n\theta\sigma'$ also are in R ∀n ∈ N. Now we will modify the time assignment τ to this family of words.

Let γ_0 be a solution that assigns to $f^{\theta}_{\varphi,\mathcal{P}}(\gamma)$ subject to $\hat{C}(\theta, \Delta)$ its maximum value. Then, by hypothesis, $f^{\theta}_{\varphi,\mathcal{P}}(\gamma_0) > 0$. We will prove that $\tau_\sigma \lhd \gamma_0 \lhd \tau_\theta \lhd \tau_{\sigma'} \in Sol(\sigma\theta\theta\sigma', \Delta)$.

We know that $\tau_\sigma \in Sol(\sigma, \Delta)$ because $\tau_\sigma \lhd \tau_\theta \lhd \tau_{\sigma'} \in Sol(\sigma\theta\sigma', \Delta)$. Then (σ, τ_σ) is a context for θ. Since θ is past-independent and γ_0 is a solution under $\hat{C}(\theta, \Delta)$ then, by corollary 9, $\tau_\sigma \lhd \gamma_0 \in Sol(\sigma\theta, \Delta)$.

The tests done in the second θ and reset in the previous iteration are satisfied since θ is past-independent. And the clocks not being reset in the first θ can be tested just by delays (because of the no-deadline constrained iterations property), and since those distances are increased they are also satisfied. Then $\tau_\sigma \lhd \gamma_0 \lhd \tau_\theta \in Sol(\sigma\theta\theta, \Delta)$.

Finally σ' has two previous iterations. If the reset is done in σ or in the first θ, the former argument of the delay holds. If the reset occurs in the second θ, we just know that it is satisfied because $\tau_\sigma \lhd \tau_\theta \lhd \tau_{\sigma'} \in Sol(\sigma\theta\sigma', \Delta)$. Thus we conclude that $\tau_\sigma \lhd \gamma_0 \lhd \tau_\theta \lhd \tau_{\sigma'} \in Sol(\sigma\theta\theta\sigma', \Delta)$.

Using the previous reasoning it follows inductively that $\tau_\sigma \lhd \gamma_0^n \lhd \tau_\theta \lhd \tau_{\sigma'} \in Sol(\sigma\theta^{n+1}\sigma', \Delta)$. Thus $(\sigma\theta^{n+1}\sigma', \tau_\sigma \lhd \gamma_0^n \lhd \tau_\theta \lhd \tau_{\sigma'}) \in L(R, \Delta)\ \forall n \in \mathrm{N}$.

It only remains to observe that the objective function could be as big as we want iterating as many times as we like.

Theorem 14 Let (R, Δ) be a strongly non-zeno and well-behaved TC-RE, let A^* be a subexpression of R such that all $\theta \in A$ are past-independent words. If the maximum value for the duration expression $f^{\theta}_{\varphi,\mathcal{P}}(\gamma)$ subject to $\hat{C}(\theta, \Delta)$ is less than or equal to zero and the test $\sim \in \{\leq, <\}$ then, by replacing A^* in R with $\epsilon \oplus A$, we get a regular expression R' for which $(R, \Delta) \models_{\mathcal{P}} \varphi$ iff $(R', \Delta) \models_{\mathcal{P}} \varphi$.

Proof 30 *It is clear that R' recognises a sub-language of R then, by Fact 3, $L(R', \Delta) \subseteq L(R, \Delta)$. So $(R, \Delta) \models_{\mathcal{P}} \varphi \Rightarrow (R', \Delta) \models_{\mathcal{P}} \varphi$.*

It remains to prove that $(R', \Delta) \models_{\mathcal{P}} \varphi \Rightarrow (R, \Delta) \models_{\mathcal{P}} \varphi$. We do it by showing that if we have two consecutives iterations of A in a timed word of (R, Δ) then we can eliminate one of them and the resulting timed word satisfies φ.

Now suppose that $(R', \Delta) \models_{\mathcal{P}} \varphi$, $(\sigma, \tau) \in L(R, \Delta)$ and $\exists i_1, i_2, i_3 \in \mathrm{N} : 0 <$ [12] $i_1 < i_2 < i_3 < |\sigma| \wedge \sigma_{[i_1, i_2-1]}$ and $\sigma_{[i_2, i_3-1]} \in A$.

Consider the timed word where the loop $\sigma_{[i_1, i_2-1]}$ is eliminated. We have eliminated symbols whose sum has a non-positive contribution because $f^{\sigma_{[i_1,i_2-1]}}_{\varphi,\mathcal{P}}(\tau_{[i_1,i_2-1]}) \leq 0$ by hypothesis. Thus it is easy to see that $f^{\sigma'}_{\varphi,\mathcal{P}}(\tau') \geq f^{\sigma}_{\varphi,\mathcal{P}}(\tau)$ where $\sigma' = \sigma_{i_1-1]}\sigma_{[i_2}$ and $\tau' = \tau_{i_1-1]} \lhd (\tau_{[i_2-1} -_{map} \tau_{i_2-1})$.

If we can prove that actually $(\sigma', \tau') \in L(R, \Delta)$, we arrive easily at the result that if the LDI is violated in the original expression it will be also violated in an expression using at most one unfold of the cycle. Therefore, let us prove that the new timed word is in the timed language.

The fusion closed property ensures that $\sigma' \in R$. And we know that $\tau'_0 = \tau_0 = 0$. So it remains to prove that $\tau' \in Sol(\sigma', \Delta)$. That is, we must check that all the constraints $(\delta, \sigma'_i, \sim c) \in \Delta$ are satisfied for all τ'_i: $\tau'_i - \tau'_{last\_\delta\_in\_\sigma'_{i-1}} \sim c$.

[12] Remember that *init* is the first transition.

First at all, any constraint is satisfied by τ'_i when $i \leq i_1 - 1$ since σ, σ' and τ, τ' coincide in the first i_1 transitions.

The other simple case is when $\delta \cap \sigma'_{[i_1,i-1]} \neq \emptyset$, the distance between τ'_i and the last δ transition in $\sigma'_{i-1]}$ is the same as the distance of the corresponding in the original τ, because the reset occurs after the eliminated iteration.

Finally we have to deal with a test after the eliminated loop that have a reset before the eliminated loop. Now let $i \geq i_1$ while $last\_\delta\_in\_\sigma'_{i-1]} \leq i_1 - 1$.

Since $last\_\delta\_in\_\sigma'_{i-1]} \leq i_1 - 1$ then $last\_\delta\_in\_\sigma'_{i-1]} = last\_\delta\_in\_\sigma'_{i_1-1]}$. But σ, σ', τ and τ' coincide in the first i_1 transitions; then $last\_\delta\_in\_\sigma'_{i_1-1]} = last\_\delta\_in\_\sigma_{i_1-1]}$ and $\tau'_{last\_\delta\_in\_\sigma'_{i_1-1]}} = \tau_{last\_\delta\_in\_\sigma_{i_1-1]}}$. Thus $\tau'_{last\_\delta\_in\_\sigma'_{i-1]}} = \tau_{last\_\delta\_in\_\sigma_{i_1-1]}}$. Therefore we have to prove that $\tau'_i - \tau_{last\_\delta\_in\_\sigma_{i_1-1]}} \sim c$.

There are two cases: (a) $i + (i_2 - i_1) \leq i_3 - 1$ and (b) $i + (i_2 - i_1) \geq i_3$.

In the case (a) we are analyzing a constraint that goes from an iteration instance to another one. Thanks to independence we can argue that the constraint is satisfied.

In the case (b) we know that δ is not reset in $\sigma_{[i_2,i_3-1]}$ because $last\_\delta\_in\_\sigma'_{i-1]} \leq i_1 - 1$ and $\sigma'_{i-1]}$ is the result of extracting $\sigma_{[i_1,i_2-1]}$ from $\sigma_{i+(i_2-i_1)-1]}$. Then we have in σ' a test after location $src(\sigma_{i_3})$ with a previous reset before i_1.

Because of the no-deadline constrained iterations property ($\sigma_{[i_2,i_3-1]}$ is a loop) the test must be a delay, i.e., $(\delta, \sigma'_i, \geq c) \in \Delta$. So we must prove that $\tau'_i - \tau_{last\_\delta\_in\_\sigma_{i_1-1]}} > c$.

Also it can be seen from the definition of $\lhd$ that $\tau'_i = \tau_{i+(i_2-i_1)} + \tau_{i_1-1} - \tau_{i_2}$. Since $\tau_{i+(i_2-i_1)} = \tau_{i_2+(i-i_1)} \geq \tau_{i_2}$ then $\tau'_i \geq \tau_{i_1-1}$. Then we get that is sufficient to prove that $\tau_{i_1-1} - \tau_{last\_\delta\_in\_\sigma_{i_1-1]}} > c$.

By the no-delay constrained iterations property we conclude that $MIN_\delta(\sigma'_{i_1-1]}) > c$. Then $MIN_\delta(\sigma_{i_1-1]}) > c$ since $\sigma_{i_1-1]} = \sigma'_{i_1-1]}$. By the definition of MIN_δ, it follows $MinTimes_\delta(\sigma_{i_1-1]}) > c$, since c is a constants appearing in a time constraint in Δ. In particular for $\tau_{i_1-1]}$ we get $\tau_{i_1-1} - \tau_{last\_\delta\_in\_\sigma_{i_1-1]}} > c$, as we need.

Chapter 10

Conclusions and Future Work

The need to predict temporal behavior of critical real-time systems has encouraged the development of an useful collection of models, results and tools for analyzing schedulability of applications (e.g., [109]). However, there is no general analytical support for verifying other kind of high-level timing requirements on complex software architectures. On the other hand, the verification of specifications and designs of real-time systems has been considered an interesting application field for automatic analysis techniques such as model-checking. Unfortunately, practicality of previous formal approaches were limited due to several factors, among them: complexity introduced by preemption modeling, and global analysis due to implicit interrelationship (the processor sharing) between "intuitively" independent components.

To cope with the challenges of formal analysis RTS Designs, we have focused on three aspects that, we believe, are fundamental to get practical tools: model-generation, model-reduction and model-checking.

From the point of view of model-generation, we have presented a formal approach to verify a class of real-time distributed designs which adhere to the hypothesis of known analytical theory for fixed-priority scheduling. With this approach in mind, we have shown how to build rather simple and conservative formal models based on timed automata to improve practicality of verification.

Following a the criteria of separation of concerns, we understand the scheduler and the priority assignment mainly as a way to achieve fair computation and predictive response times. Then, we abstract away the scheduler behavior including preemptiviness by using the WCCT and the resulting formal models could be understood as virtually running on dedicate processors. CDAGs describe potential sequences of task events and WCCT-BCCT provides the bound for their occurrences. In our application models, we require that tasks should not suspend themselves, a common assumption for schedulability analysis. This rules out, for instance, general handshake synchronization. However, communication is achieved through shared protected objects, and signals. Mutual exclusion is achieved

through Priority Ceiling Protocol emulation. This is another important property provided by the scheduler and captured by our modeling technique.

Then, this "asynchronous" application model for RTS greatly simplifies WCCT calculus and some researches found reasonable for many applications. Shared objects are non-blocking from a semantics point of view and a bounded blocking time occurs just to achieve mutual exclusion. Thus, the TA edges standing for the end of operations on shared objects are guarded with timing conditions that take into account blocking time due to mutual exclusion. Therefore, we are able to build abstractions to automatically reason on response times. We also believe that, in most cases, this is the kind of reasoning that a designer would rely on to gain confidence on its design. For instance, in our models two events standing for the end of mutual exclusive operations may be arbitrarily close (if allowed by timing conditions). Of course, this does not happen in the implementation but we believe that mutual exclusion is just a way to achieve "serializable reentrant" operations and that is what we actually model. That is, we believe that there are only two important properties that must be captured about mutual exclusion (a) operations work properly (b) there is a timing impact on tasks, the blocking time.

Clearly, it may be argued that our scheduler abstraction may not be able to verify properties that hold due to subtle scheduling side-effects. That is the price we pay. The properties we are able to check are properties which satisfaction just depends on response times and do not depend in any other characteristic on the underlying scheduling discipline. This could be an informal way to understand the scope and limitations of our approach: a way to automatically reason on worst case behavior of a set of tasks checking properties which are independent from any other aspect of the scheduler but the achieved response times. Our experiments have shown that we could check reasonable estimations. For a more detailed analysis for scheduling dependent, we would recommend another more accurate but less feasible approach. One example of those lost properties, which might be expected by the designers, is task precedence produced by harmonic periods and a clever assignment of priorities [77]. In the future work section we mention the possibility of filtering the behavior with some rules that would hold in the actual implementation.

To a certain extent, our proposal enlarges the applicability of scheduling theory to prove requirements which involve interaction among components. As another contribution, we have also shown how known scheduling results could help in reducing the complexity of models and their analysis (by making them more compositional). To model such systems, we present I/O Timed Components, a simple notion build on top of Timed Automata to get live non-blocking models, also providing some important methodological advantages like influence detection (Chapter 7), compositional reasoning [111], etc.

We have also provided a set of mechanisms to abstract the model according to the observation power needed to verify the requirement. Thus, we do not feed the verification back-end with the whole model as previous formal approaches. To provide arguments about the correctness of those abstractions, we have developed a notion of continuous observational bismulation that is weaker than strong timed bisimulation yet preserving TCTL logic.

Furthermore, by choosing timed automata, we have adapted and applied their deeply studied and developed analysis theory. Consequently, model-checking can be performed by the well-known automatic analysis tools for Timed Automata ([61, 23, 91, 146], etc.), which were successfully applied to several case studies [54, 59, 62, 148, 147],etc. Moreover, we have also described from scratch an algorithm to model-check duration properties, a feature that is not addressed by available tools.

We have implemented prototype versions of some of the components of the described tool. They have served to experimentally validate the approach. The combination of the explained modeling approach with the reduction methods has shown to be very effective for checking middle size examples found in literature like the Mine Pump design [40] and the active structure system [69]. We believe in the scalability of the method to big-size examples due to the ability of our method to smartly focus on a submodel which is relevant to the requirement to be checked. On the other hand, parallel composition reduction by relevance (Chapter7) has achieved dramatic speedups in several examples beyond the design models proposed in this thesis.

Besides getting an deliverable and integrated version of the whole tool, there are at least three directions to enhance the approach: modeling aspects, model manipulation, and model-checking.

Modeling Issues Further research in combining both scheduling theory and formal analysis seems highly recommendable. That is, the fundamental modeling idea presented in this thesis can be summarized as the use of calculated response times to build a conservative, compositional and simple formal model in order to check complex requirements. That idea was applied to the Fixed-Priority Application model but we believe it is possible to generalize the proposed method to cope with a broader set of run-time scheduling disciplines like EDF (Earliest Deadline First) [43], etc. We also think that it is possible to use these ideas to analyze systems which processing nodes are scheduled using different disciplines (e.g., processing nodes scheduled at pre-run time, processing nodes with non-preemptive aperiodic tasks [70]).

In future, we may be interested in analyzing designs of applications models with no underlying analytical scheduling theory. For instance, imagine a cooperative scheduling where each task is composed by a sequence of non-preemptable subtasks and event arrival can not characterized as cyclic or sporadic due to jitter, etc [70, 15]). In those cases, we may use an adequate formal model just to obtain the maximum and minimum response times. Then, it could be built another compositional, conservative and less detailed model -like the one presented in this thesis- to analyze complex requirements. Particularly - to obtain those maximum and minimum response times-, we can resort to hybrid automata for models like [15], symbolic model checking for models like [46], or reachability graph [104] on models like [70].

On the other hand, the method can be extended to support more features and phenomena such as static and dynamic offsets for task release, task jitter [131], mode change (i.e., a

reconfiguration of the set of tasks running), synchronous communication among tasks, etc. For instance, it is not difficult to add an "enabling-disabling" mechanism to model changes of mode. Tasks are initially enabled but they can be specified to be disabled. A task can be enabled by another task using the command *enable* applied to a set of task names. The language could provide special events *Enable*+TaskName to enable tasks. Intuitively, if an enabled task receives an enabling signal it just ignores it. If a task receives that signal while disabled it goes to an enabled state and behaves as normal. It is important to say that the time elapsed while disabled is taken into account when it becomes enabled again. That is, if the period has elapsed while disabled a periodic task is released immediately when enabled again. Similarly, if minimal interarrival time has elapsed a sporadic task will be able to serve a signal immediately after being enabled. A task can disabled itself at the end of its code. The enabling disabling graph over the tasks could be used to statically find out whether two tasks could be active at the same time and thus use this information for the WCCT calculus. It is worth mentioning these mechanisms could be used to perform remote calls (a task performs a call, it enables a task that should receive the answer and disables itself).

More accuracy could be gained by filtering out behaviors of final models. That is, it is possible to add observer automata to filter out some impossible temporal behaviors (using some "impossible" trap state). For instance, it could be filtered out impossible behaviors resulting from facts like precedence given by task priorities with harmonic periods [77]. It is worth warning that filtering should satisfy some kind of non-zeno property, namely, any non-rejected finite run (i.e. a run not arriving to trap state that stands for "impossible" behavior) could be extended to a (non-rejected) divergent run.

Another "software engineering-line of research is the definition of a more user-friendly "pattern language" to replace the direct use of TA to express generic scenarios (e.g. [67]) and to display out counterexamples (e.g.[95]).

A different line of research is the parametric modeling of Real-Time Systems ([8]). That is, some parameters of the system (e.g., the response time of a task) could remain unknown and the analysis may provide constraints on them. Several lines of research could emerge from it, namely, synthesis of maximum response times to met a requirement (for instance, to find out local deadlines during design), measuring the criticality of response times to meet a requirement (for example, to used in testing or reengineering activities), etc.

Model Manipulation The engineering focus of this thesis has not required us to study and solve some interesting theoretical open questions with respect to CO-bismilarity. However, it is worth stating them as future work, namely, is CO-bisimilarity decidable?; are there two models satisfying the same set of TCTL formulas which are not CO-bsismilar models?, etc.

Our reduction method for the parallel composition could be improved to deal specially with the notion of relevant clocks or relevant discrete variables to perform a more precise analysis.

To cope with requirements involving large subsystems some extra abstraction mechanism may be useful. The idea is to provide automatic support for checking that an abstraction relation really holds (see for example [101]). On the one hand, we believe that we can check correspondence of our I/O timed components against specifications given in terms of Timed Automata by using the homomorphism technique under an "assume/guarantee" framework presented in [111]. On the other hand, it seems relatively easy to check abstractions for I/O Timed Component (similar ideas can be found in [101]). It is well-known that non-Deterministic Timed Automata are not closed under complementation [2]. However, deterministic components (components which underlying TA is deterministic) can be complemented and therefore language inclusion can be checked (building the parallel composition $A \parallel \neg B$.). Given an alphabet Σ, the complementation is done by: (a) adding a trap location stuttering all the elements of Σ (b) for each location and for each label $l \in \Sigma$, add a transition with label l to the trap location with a guard that negates the disjunction of the guards of original transitions with that location as source, (c) eliminate the invariants and add a transition with invariant negated to the trap location. The language that remains in the trap location is the complement of the language of the component. Thus, we can imagine a technique to check abstractions provided by the user as deterministic I/O components. It is just required to check whether there is a run leading to the trap location in the parallel composition of the component that want to be replaced and the complemented abstraction. As an example, in a fault tolerant version of an active structure control system [69], suppose that three sensors communicate with a voter which in turn communicates with the modeler. We can replace the three sensors, the 4 handshake communication mechanism and the voter with a component that behaves like a virtual sensor that needs within 20 and 90 t.u. to complete the sensing and start the communication with the modeler (see Fig. 10.1).

Another rewarding effort may be a change of kernel formalism to analyze a discrete-time version of I/O timed components. For example, they can be translated into temporal process algebra with a maximal progress assumption. Then, we would be able to apply tool support of "component-wise" minimization like in [69].

Model Checking Although this thesis was not focused on model-checking technologies there might be a substantial number of improvements for these kind of applications. For instance, the topology of the composition of a safety observer with the system turns out into a non-trivial DAG of Strongly Connected Components (SCC). Model-Checking tools like KRONOS are based on a fixed point calculation on a symbolic state representation. Currently we are working to improve the fixed point calculation for these topologies (following the DAG structure).

The next step for the checking duration properties is the development of a tool. New results to enlarge the class of accurate analysis could be also be studied. As future work, we would like to adapt some recently presented optimization techniques to our framework (e.g. [114, 22]). In particular, we believe that partial order techniques (see [22]) can be adapted to check linear duration invariants that depends on the locations of just one

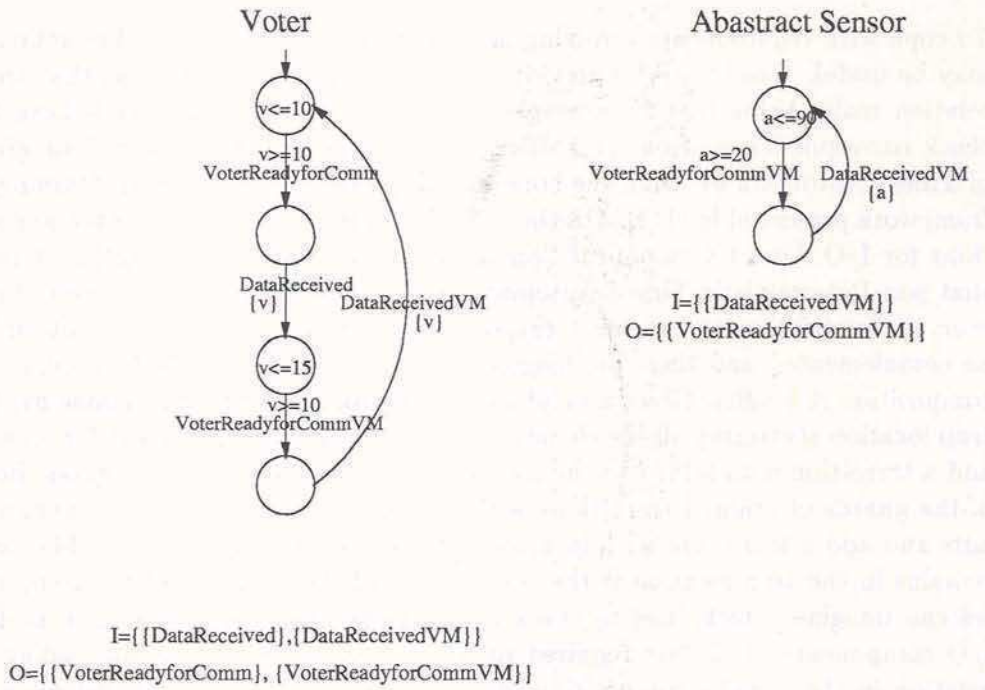


Figure 10.1: The Voter and an Abstraction for the Fault Tolerant Sensor

automaton of a parallel composition. Another interesting topic of research is the study of on the fly techniques to avoid the construction of the whole reachability graph (unfolding the necessary subexpressions), making some kind of lazy evaluation of the composed phases of the algorithm mixed with some dynamic programming (like in the Kleene calculus). We also would like to analyze whether other post-stable symbolic state space representation, like [149], are suitable for our application. On the other hand, the construction of the regular expression is also a key issue. The order in which nodes of the Reachability Graph are considered (e.g.,: depth first), might have an impact on the size of resulting Regular Expressions. Reduction rules might also be very helpful to keep the expression compact. On the other hand, the analysis over the Regular Expressions can be done symbolically by unfolding the Kleene definition by demand. Many mixed strategies are possible and should be studied. Another idea is to detect particular nodes of the graph that might help in reducing the size of the final result. For example, suppose that we have detected locations of the Reachability Graph such that all sequence starting from them are past independent (Def. 48 in Chapter 9), the "milestones" (we believe that a subset of them can be detected easily over the input timed automaton). Then, the maximum or minimum for all paths of $R^k_{i,j}$ with i, j milestones can be calculated locally and that value can be stored instead of the paths themselves.

Although we did several experiments on some real-life medium-size problems found in literature we believe that only the accumulation of experiments coupled with the fine-tuning of tools will give the final word on the practical value of this approach.

Bibliography

[1] B. Alpern and F. Schneider, "Verifying Temporal Properties without Temporal Logic," *ACM Trans. Programming Languages and Systems*, Vol. 11, No. 1, 1989, pp. 147-167.

[2] R. Alur, *Techniques for Automatic Verification of Real-Time Systems,* doctoral disertation, Stanford University, 1991.

[3] R. Alur, C. Courcoubetis, and D. Dill, "Model-Checking for Real-Time Systems," In *Proceedings of Logic in Computer Science*, IEEE Computer Society, Los Alamitos, Calif, pp. 414-425, 1990. Also in *Information and Computation,*, vol. 104, no. 1, pp.2-34, 1993.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi, "An implementation of Three Algorithms for Timing Verification based on Automata Emptiness," In *Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[5] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The Algorithmic Analysis of Hybrid Systems," *Theoretical Computer Science*, vol. 138, 1995, pp.3-34.

[6] R. Alur, C. Courcoubetis, and T.A. Henzinger, "Computing Accumulated Delays in Real-Time Systems," In *Proceedings of the 5th International Conference on Computer Aided Verification, CAV'93*, Lecture Notes in Computer Science 697, pages 181–193. Springer-Verlag, 1993.

[7] R. Alur and D. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, 1994, pp. 183-235.

[8] R. Alur, T. A. Henzinger, and M. Vardi, "Parametric Real-Time Reasoning," *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC 1993)*, pp. 592-601.

[9] E. Asarin, O. Maler, and P. Caspi, "A Kleene Theorem for Timed Automata", In G. Winskel, editor, *Logics In Computer Science*, 1997.

[10] E. Ashcroft and Z.Manna, "Formalization of Properties of Parallel Programs," *Machine Intelligence*, vol. 6, pp. 17-41, 1971.

[11] N.C. Audsley, *Flexible Scheduling of Hard Real-Time Systems,* Dissertation Thesis, Department Computer Science, University of York, UK, August 1993.

[12] N.C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling", *Software Engineering Journal*, Vol. 8, No. 5, Sept. 1993, pp. 284-292.

[13] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings, "STRESS: A Simulator For Hard Real-Time Systems," *Software Practice and Experience*, 1994.

[14] G.S. Avrunin and J.C. Corbett. "A Practical Technique for Bounding the Time Between Events in Concurrent Real Time Systems," *Proceedings of the 1993 International Symposium on Software Testing and Analysis,* Cambridge MA, pp. 110-116, June 1993. also published in Software Engineering- Notes (18:3), July 1993.

[15] G.S. Avrunin, J.C. Corbett, and L.K. Dillon, "Analyzing Partially Implemented Real-Time Systems," *IEEE Trans. Software Eng.,*, Vol. 24, No. 8, Aug. 1998.

[16] G.S. Avrunin, J.C. Corbett, L.K. Dillon, and J.C. Weilden, "Automated Derivation of Time Bounds in Uniprocessor Concurrent Systems," *Transaction on Software Engineering*, vol. 20, no. 9, September 1994.

[17] J.C.M. Baetan, and W.P. Weijland, *Process Algebra,* Cambridge University press, 1990.

[18] T.P. Baker, "A Stack Based Allocation Policy for Real-Time Processes," *Proceedings of the 11th Real Time Systems Symposium*, pp 191-200, December 1990.

[19] L. Baresi, A. Orso, and M. Pezze, "Introducing Formal Specification Methods in Industrial Practice," *Proc. IEEE International Conference On Software Engineering*, 1997.

[20] C. Bellettini, M. Felder and M. Pezzè, "Merlot: A tool for analysis of Real-Time Specifications," *Proc. of 7th International Workshop on Software Specification and Design*, 1993.

[21] H. Ben-Abdallah, Y. Si Kim, and I. Lee, "Schedulability and Safety Analysis in the Graphical Communicating Shared Resources," *Proc. of IEEE WORDS'96: 2nd Workshop on Object Oriented Real-Time Dependable Systems*, Feb. 1996.

[22] J Bengtsson, B Jonsson, J Lilius, and Wang Yi, "Partial Order Reductions for Timed Systems," In *Proc. of CONCUR'98*, September 1998.

[23] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL- A Tool Suite for the Automatic Verification of Real-Time Systems," *Proc. Hybrid Systems III*, Lecture Notes on Computer Science 1066, Springer Verlag, 1996, pp. 232-243.

[24] R. Bettati, J.W.S. Liu, "End-to-End Scheduling to Meet Deadlines in Distributed Systems," *Proceedindgs of the 12th International Conference on Distributed Computing Systems*, Yokohama, Japan, Jun. 1992.

[25] B. Berthomieu, M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions On Software Engineering*, vol. 17, no. 3, March 1991.

[26] S. Bornot, J. Sifakis. An Algebraic Framework for Urgency To appear *in Information and Computation*, Academic Press, 2000.

[27] V. Braberman, "Integrating Scheduling Theory into Formal Models of Real-Time Systems," *Proc. KIT 125 workshop,* Como, Italy, Sept. 1997.

[28] V. Braberman and Dang Van Hung, "On Checking Timed Automata for Linear Duration Invariants," *Proc.IEEE 1998 Real Time Systems Symposium,* IEEE Computer Soc. Press, Los Alamitos, Calif., Dec. 1998, pp. 264-273.

[29] V. Braberman, Dang Van Hung. "On checking Timed Automata for Linear Duration Invariants," *TR 135, UNU-IIST,* February 1998.

[30] V. Braberman, M. Felder, "Verification of Real-Time Designs: Combining Scheduling Theory with Automatic Formal Verification," *Proc. 7th European Conf. on Software Eng./ 7th ACM SIGSOFT Symposium on the Foundations of Software Eng., (ESEC/FSE 99),* Lecture Notes in Computer Science 1687, Springer Verlag, Sept. 1999, pp.494-510.

[31] S. Bracley, W. Henderson, D.Kendall, and A. Robson, "Designing and Implementing Correct RTS," Technical Report University of Northumbria at Newcastle.

[32] P. Bremond-Gregoire, and I. Lee, "A Process Algebra of Communicating Shared Resources with Dense Time and Priorities," Technical Report, University of Pennsylvania, MS-CIS-95-08.

[33] G. Bucci, M. Campanai, P. Nesi, "Tools for specifying Real-Time Systems," *Real Time Systems Journal*, vol. 8, issue 2/3, Kluwer Academic Publisher, March/May 1995.

[34] G. Bucci, and E. Vicario, "Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets," *IEEE Transactions On Software Engineering*, vol. 21, no. 12, December 1995.

[35] T. Bultan, J. Fisher, and R. Gerber, "Compositional Verification by Model Checking for Counter Examples," *Proc. of the International Symposium on Software Testing and Analysis*, 1996.

[36] J.R. Burch, E.M. Clarke, K.L. McMillian, and D.L. Dill, "Symbolic Model Checking. 10^20 states and beyond," *Proc. 5th IEEE Symp. Logic in Computer Science,* 1990, pp. 428-439.

[37] A. Burgueno, and F. Boniol, "Un Modelo de Sitemas Hibridos Orientado a la planificacion de procesos concurrentes," *Proc. of the II Jornadas de Informatica*, pages 585-594, Almucar, Granada, pp. 15-19, July 1996.

[38] A. Burgueno and V. Rusu, "Task-system Analysis Using Slope-Parametric Hybrid Automata," *Euro-Par'97 Workshop on Real-Time Systems and Constraints*, Passau, Germany, Aug. 1997.

[39] A. Burns, "Preemptive Priority Based Scheduling: An Appropriate Engineering Approach," Technical Report, York University, YCS-93-214.

[40] A. Burns and A. Wellings, *Real-Time Systems: Specification, Verification and Analysis,* Prentice Hall, 1996.

[41] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages,* Addison-Wesley, 1996.

[42] A. Burns and A. Wellings, *HRT-HOOD: A Structured Design Method for Hard Real-Time ADA Systems,* Elsevier, 1995.

[43] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications,* Kluwer Academic Publishers, Boston, 1997.

[44] J. Büchi, "On a Decision Method in Restricted Second-Order Arithmetic," *Proc. Int. Congress on Logic, Methodology, and Philosophy of science 1960,* pp. 1-12. Standford University press, 1962.

[45] S. Campos, "The Priority Inversion Problem and Real Time Symbolic Model Checking," Technical Report, Carnegie Mellon University, CMU-CS-93-125.

[46] S. Campos, E. Clarke, W. Marrero and M. Minea, "VERUS: A Tool for Quantitative Analysis of Finite State Real-Time Systems," *Proc. of SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems,* 1995.

[47] S. Campos and O. Grumberg, "Selective Quantitative Analysis and Interval Model Checking: Verifying Different Facets of a System," *Proc. of Int'l Conf. Computer Aided Verification*, Lecture Notes on Computer Science 1102, Springer Verlag, 1996, pp.257-268.

[48] R. Cleaveland, S.A. Smolka, O. Sokolosky, "The Concurrency Factory: A development Environment for Concurrent Systems," *Proc. of Computer Aided Verification 1996.*

[49] P. Chan, and D. Van Hung, "Duration Calculus Specification for Tasks with Shared Resources," *Proc. of Algorithms, Concurrency and Knowledge (Asian Computer Science Conference)*, Pathumthani, Thailand, December 1995. Lecture Notes in Computer Science 1023, Springer Verlag.

[50] S. Chay Cheng, J. Stancovic, and K. Ramamritham, "Scheduling Algorithms for Hard Real Time Systems: A Brief Survey," 1987.

[51] M.I. Chen, *Schedulability Analysis of Resource Access Control Protocols in Real Time Systems,* Ph.D. thesis, University of Illinois at Urbana-Champaign, 1991.

[52] S.C. Cheung and J. Kramer, "Checking Safety Properties Using Compositional Reachability Analysis," *Trans. on Software Eng. and Methodology*, Vol. 8, No. 1, Jan. 1999, pp. 49-79.

[53] E. Clarke and E.A. Emerson, "Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic," *Workshop on Logic Programs*, Lecture Notes on Computer Science 131, 1981.

[54] E. Clarke and J. Wing, "Formal Methods: State of the Art and Future Directions," *ACM Computer Surveys*, Vol. 28, No. 4, pp. 623-643, December 1996.

[55] E. Clarke, O. Grumberg and D. Long, "Model Checking and Abstraction," *Proc., Principles of Programming Languages (POPL)*, 1994.

[56] E. Clarke, O. Grumberg and D. Peled, *Model Checking,* MIT Press, January 2000.

[57] D. Clarke, I. Lee and H.L. Xie, "VERSA: A Tool for the Specification and Analysis of Resource-Bound Real-Time Systems" *Journal of Computer Software Eng.*, Vol. 3, No. 2, 1995.

[58] J.C. Corbett, "Timing Analysis of ADA Tasking Programs," *IEEE Transaction On Software Eng.*, Vol. 22, No. 7, Jul. 1996, pp. 461-483.

[59] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans, "The bounded retransmission protocol must be on time!," *Proc. of the 3rd Int'l. Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Lecture Notes in Computer Science 1217, Springer Verlag. 1997, pp. 416-431.

[60] C. Daws, "Verification de Systemes Temporises par Resolution de Systemes de Contraintes Lineaires," *In Memoire de DEA*, Institut Nationale Polytechnique de Grenoble, June 1993.

[61] C. Daws, A. Olivero, S. Tripakis and S. Yovine, "The Tool KRONOS," *In Proc. of Hybrid Systems III*, LNCS 1066, Springer Verlag, 1996, pp. 208-219.

[62] C. Daws, S. Yovine, "Two examples of verification of multirate timed automata with KRONOS," *Proc. of the 1995 IEEE Real-Time Systems Symposium, RTSS'95*, Dec.1995. IEEE Computer Society Press.

[63] C. Daws and S.Yovine, "Reducing the Number of Clock Variables of Timed Automata," *Proc. IEEE Real-Time Systems Symposium '96*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996.

[64] Dang Van Hung and Pham Hong Thai, "On Checking Parallel Real-Time Systems for Linear Duration Invariants," *Proc. of the Inter. Symposium on Software Engineering*

for Parallel and Distributed Systems, IEEE Computer Society Press, pages 61–71, 1998.

[65] R. De Nicolla, U. Montanari, and F.Vaandrager, "Back and Forth Bisimulations," *Proc. CONCUR '90*, Amsterdam, LNCS 458, Springer Verlag, pp.152–165, 1990.

[66] B.P. Douglass, *"Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns,"* Addison Wesley, Object Technology Series, 1999.

[67] M. Dwyer, G. Avrunin, and J. Corbett, "Patterns in Property Specifications for Finite-state Verification ," *Proc. of the 21st Intl. Conference on Software Engineering*, May, 1999.

[68] M. Dwyer and C. Pasareanu, "Filter-Based Model Checking of Partial Systems," *Proc. ACM SIGSOFT/FSE*, November, 1998.

[69] W. Elseaidy, R. Cleaveland, and J. Baugh Jr., "Modeling and Verifying Active Structural Control Systems, " *Science of Computer Programming,* 29(1-2):99-122, July 1997.

[70] C. Ericsson, A. Wall, and Wang Yi, "Timed Automata as Task Models for Event-Driven Systems," *Proc. of RTSCA'99*, IEEE Computer Soc. Press, Los Alamitos, Calif., Dec 1999.

[71] M. Felder and M. Pezzè, "A Formal Approach to the Design of Real-Time Systems," *WorkShop KIT125*, September 1997, pp. 23-56.

[72] C. Fidge, M. Utting, P. Kearney, and I. Hayes "Integrating Real-Time Scheduling Theory and Program Refinement," *Proc. Formal Methods Europe,* March 1996, Lecture Notes in Computer Science 1051, Springer Verlag.

[73] A.N. Fredette and R. Cleaveland, "RTSL: A Formal Language for Real-Time Schedulability Analysis," *Proc. IEEE Real-Time Systems Symposium,* , December 1993, pp. 274-283.

[74] D. Garbervetski, *Un Método de Reducción para la Composición de Sistemas Temporizados,* Master Thesis, Universidad de Buenos Aires, 2000.

[75] D. Gaudreau, and P. Freedman, "Temporal Analysis and Object-Oriented Real-Time Software Development: a Case Study with ROOM/ObjecTime," *Proceedings of IEEE Real-Time Technologies and Applications Symposium*, Boston, June 10-12, 1996.

[76] R. Gawlick, R. Segala, J. Sogaard-Andersen, N. Lynch "Liveness in Timed and Untimed Systems," *Proceedings of Int. Conference on Automata, Languages, and Programming,* , Lecture Notes in Computer Science 820, Springer Verlag, pp. 166-177, 1994. Also in, *Information and Computation*, March 1998.

[77] R. Gerber, S. Hong and M. Saksena, "Guaranteeing Real-Time Requirements with Resource-Based Calibration of Periodic Process," *IEEE Transaction On Software Eng.*, Vol. 21, no. 7, Jul. 1995.

[78] R. Gerber, W. Pugh and M. Saksena, "Prametric Dispatching of Hard Real-Time Tasks," *IEEE Transaction On Computers*, Vol. 44, no. 3, March 1995.

[79] R. Gerber and S. Hong, "Compiling Real-Time Programs with Timing Constraint Refinement and Strcutural Code Motion," *IEEE Transaction On Software. Eng.*, Vol. 21, no. 5, May 1995.

[80] C. Ghezzi, D. Mandriolli, S. Morasca, and M. Pezzè, "A Unified High-Level Petri Net Formalism for Time Critical Systems," *IEEE Transaction On Software Eng.*, Vol. 17, No. 2, Feb. 1991, pp.160-172.

[81] R. Van Glabbeek, W. Weijland, "Branching Time and Abstraction in Bisimulation Semantics (extended abstract)," *Information Processing 89, Proc. IFIP 11th World Computer Congress*, San Francisco, North Holland, pp.613-618, 1989.

[82] H. Gomaa, *Software Design Methods For Concurrent and Real-Time Systems* SEI Series in Software Engineering. 1994.

[83] S. Graff, C. Loiseaux, "A Tool for Symbolic Program Verification and Abstraction," *Proc. Computer Aided Verification*, 1993, pp.71-84.

[84] N.Halwachs, F.Lagnier, and P.Raymond, "Synchronous Observers and the Verification of Reactive Systems," *Intl. Conf. on Algebraic Methods and Software Technology*, AMAST'93, Workshops in Computing, Springer Verlag, 1993.

[85] M. Hansen, "Model-checking Discrete Duration Calculus," *Formal Aspects of Computing*, 6(6A):826–846, Nov-Dec 1994.

[86] M.G. Harbour, M.H. Klein, and J.P. Lehoczky, "Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems," *IEEE Transaction on Software Eng.*, Vol. 20, No. 1, Jan. 1994, pp.13-28.

[87] D. Hatley and I. Pirbhai, *Strategies for Real-Time System Specification,* Dorset House. New York. 1987.

[88] C. Heitmeyer, D. Mandrioli, *Formal Methods for Real-Time Computing,* John Wiley & Sons, 1996.

[89] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj, "Using Abstractions and Model Checking to Detect Safety Violations in Requirements Specifications," *Transactions on Software Eng.*, Vol.24, No.11, Nov. 1998, pp. 927-948.

[90] T.A. Henzinger, "Sooner is Safer than Later," *Information Processing Letters*, Vol. 43, 1992, pp. 135-141.

[91] T.A. Henzinger, P.H. Ho and H Wong-Toi, "A user guide to HyTech," *Proc. of Int'l. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, Lecture Notes in Computer Science 1019, Springer Verlag, 1995, pp. 41-71.

[92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems." *Information and Computation*, 111(2), pp.193-244, 1994. Special issue for LICS' 92.

[93] M.R. Hill and M. Joseph, "Automated Timing Analysis of Real-Time Prototype," *Software Engineering Journal,* Sept. 1994, pp. 221-227.

[94] C.A.R. Hoare, *Communicating Sequential Processes,* Prentice Hall, 1985.

[95] G. Holzmann, "The Model Checker Spin," *IEEE Trans. on Software Engineering*, Vol. 23, 5, pp. 279-295, May 1997.

[96] J.E. Hopcroft and J.D. Ulman. *Introduction to Automata Theory, Languages and Computation,* Adison-Wesley, 1979.

[97] M. Humprey and J. Stankovic, "CAISARTS: A Tool for Real-Time Scheduling Assistance," *Proc. IEEE 1995 ReaTime System Symposium*, IEEE Computer Soc. Press, Los Alamitos, Calif, 1995.

[98] Introspect Technologies, Inc., Colorado Springs, CO. iRAT. Technical Overview. Introspect Technologies, Inc., Colorado Springs, CO., 1994.

[99] F. Jahanian and D. Stuart, "A Method for Verifying Properties of Modechart Specifications," In *Proc. of the 9th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, Los Alamitos, Calif., 1988.

[100] M. Jackson, *"Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices"* Addison Wesley, ACM press, 1995.

[101] H. Jensen, K.G. Larsen, and A. Skou, "Scaling up UPPAAL: Automatic Verification of Real-Time Systems using Compositionality and Abstraction," In *Proc. of FTRTFT'00*,Springer-Verlag, 2000.

[102] *Real Time Systems: Specification, Verification and Analysis,* M. Joseph, ed., Prentice Hall, 1996.

[103] M. Jourdan and F.Maraninchi, "Static Timing Analysis of Real-Time Systems," ACM SIGPLAN 95.

[104] I. Kang, I. Lee, and Y.S. Kim, "An Efficient Space Generation for the Analysis of Real-Time Systems". In *Proceedings of the International Symposium on Software Testing and Analysis*, 1996. Also in *Trans. on Software Engineering*, 1999.

[105] H. Kwak, I. Lee, A. Philippou, J. Choi, and O. Sokolsky, "Symbolic Schedulability Analysis of Real-Time Systems," Proceedings of the 1998 IEEE Real Time Systems Symposium, IEEE Computer Society Press, 1998.

[106] Y. Kesten, A. Pnueli, J. Sifakis, S. Yovine, Integration Graphs: A Class of Decidable Hybrid Systems. In *Proceedings of Workshop on Theory of Hybrid Systems*, volume 736, pages 179–208. Springer-Verlag, June 1992. Also in *In Information and Computation,* Academic Press, 150(2), pages 209–243, 1999.

[107] M.H. Klein, J. Goodenogh, and L. Sha, "Rate Monotonic Analisys for Real Time Systems," Technical report SEI/CMU. TR91-006. 1991.

[108] M.H. Klein, J.P. Lehoczky, and R. Rajkumar, "Rate Monotonic Analysis for Real-Time Industrial Computing," *IEEE Computer,* Jan. 1994, pp. 24-32.

[109] M.H. Klein, T. Ralya, P. Pollak, R. Obenza, and M.G. Harbour, *A Practitioner's Handbook for Real-Time Analysis - Guide To Rate Monotonic Analysis for Real Time Systems,* Software Engineering Institute ed., Kluwer academic Publishers, 1993.

[110] R. P. Kurshan, *"Computer-Aided Verification of Coordinating Processes The Automata-Theoretic Approach",* Princeton University Press, 1995.

[111] R. P. Kurshan S. Tasiran, R. Alur, and R. K. Brayton, "Verifying Abstractions of Timed Systems", In *Proc. of the 7th Int'l. Conf. on Concurrency Theory (CONCUR 1996)*, Lecture Notes in Computer Science 1119, Springer Verlag, 1996.

[112] W. Lam and R. Brayton, "Criteria for the Simple Path Property in Timed Automata," In *Proceedings of the 6th International Conference on Computer Aided Verification, CAV'94*, number 818 in Lecture Notes in Computer Science, pages 27–40. Springer-Verlag, June 1994.

[113] W. Lam and R.K. Brayton, "Alternating rq Timed Automata," In C. Courcoubetis, editor, *Proceedings of the 5th Inter. Conference on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, pages 236–252. Springer-Verlag, June/July 1993.

[114] K. Larsen, F.Larsson, P. Pettersson, and Wang Yi, "Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction," In *Proc. of the 18th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, Los Alamitos, Calif., December 1997.

[115] K.G.Larsen and Wang Yi, "Time Abstracted Bisimulation: Implicit Specifications and Decidability," *Information and Computation*,Academic Press, 134, 75-101, 1997.

[116] M. Lawford and W.M. Wonham, "Equivalence Preserving Transformations for Timed Transition Models," *IEEE Trans. on Automatic Control* Vol. 40, No. 7, July 1995, pp. 1167-1179.

[117] I. Lee, P. Bremond-Gregoire and R. Gerber, "A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems," *Proc. of the IEEE, Special Issue on Real-Time Systems*, Jan. 1994, pp. 158-171.

[118] J.P. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotone Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proc. of the 10th Real-Time Systems Symposium*, IEEE Computer Society Press, December 1989.

[119] Z. Liu, M. Joseph, and T. Janowski, "Verification of Schedulability for Real-Time Programs," *Formal Aspects of Computing, 1995.*

[120] Z. Liu and M. Joseph, "Specification and Verification of Fault Tolerance, Timing and Scheduling," *Transaction On Programming Languages and Systems*, vol. 21, no. 1, pp:46-89, January 1999

[121] C. L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in Hard Real Time Environment," *Journal of the Association of the Computing Machinery*, vol. 20, No 1, pp-46-61, January 1973.

[122] J.W.S. Liu, J.L. Redondo, Z. Deng, T.S. Tia, W. Shih, and R. Beattati, "PERTS: A Prototyping Environment for Real-Time Systems," *In Proc. IEEE 1993 Real Time Systems Symposium*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1993.

[123] J. McManis and P. Varaiya, "Suspension Automata: A Decidable Class of Hybrid Automata," *Proc. of CAV 1993.*

[124] P. Merlin and D.J.Farber, "Recoverability of Communicating Protocols," *IEEE Transactions on Communications*, vol. 24, no. 6, September 1976.

[125] M. Merritt, F. Modugno, and M. Tuttle, "Time Constrained Automata," *Proc. CONCUR'91*, LNCS, Springer Verlag, vol. 527, 1991.

[126] R. Milner, *"Communication and Concurrency,"*. Prentice Hall, 1989.

[127] X. Nicollin, J-L. Richier, J. Sifakis, and J. Voiron, "ATP: an Algebra for Timed Processes," *IFIP TC 2*,1990.

[128] X. Nicollin, J. Sifakis, and S. Jovine, "Compiling Real-Time Specification into Extended Automata," *IEEE Trans. on Soft. Eng.*,Specila Issue on Real-Time Systems, Vol. 18, 9, pp. 794-804, September 1992.

[129] A. Olivero, J. Sifakis, and S. Yovine, "Using Abstractions for Validation of Linear Hybrid Systems," *Proc. of 6th Computer-Aided Verification*, pages 81–94, California, July 1994. Lecture Notes in Computer Science 818, Springer-Verlag.

[130] J.S. Ostroff, "Deciding Properties of Timed Transition Models," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 170-183, 1990.

[131] J.C. Palencia and M.G. Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," *Proc. of the Real-Time Systems Symposium*, Dec. 1998, pp. 26-39.

[132] A. Parashkevov and J. Yantchev, "ARC - A Verification Tool for Concurrent Systems," *Proc. of the Third Australasian Parallel and Real-Time Conference*, Brisbane, Australia, Sep. 1996.

[133] *IEEE POSIX.4. Real-Time Extensions for Portable Operating Systems,* IEEE Computer Society Press. 1992.

[134] R.L. Rajkumar, l. Sha, and J.P. Lehoczky, "Real Time Synchronization of Multiprocessors," *Proc. 9th Real-Time Systems Symposium*, Dec. 1988, pp. 259-269.

[135] Requiremenst Group for for Real-Time Extensions for Java Platform, *Requirements for Real-Time Extensions for Java Platform,* National Institute of Standards and Technology, http://www.nist.gov/rt-java. Apr. 1999.

[136] Y. Ripoll, A. Crespo, and A.Mok, "Improvement in Feasibility Testing for Real Time Tasks," *Real Time Systems Journal*, vol. 9, no. 2, Kluwer, September 1995.

[137] G.Reed and A. Roscoe, "A Tikmed Model for Communicating Sequential Processes," *Theoretical Computer Science*, vol. 58, pp. 249-261, 1998.

[138] , T. Rus, C. Rattray *Theories and Experiences for Real-Time System Development,* World Scientific Publishing Company, Inc., 1995.

[139] M. Saksena, A. Ptack, P. Freedman, and P. Rodziewics, "Schedulability Analysis for Automated Implementations of Real-Time Object-Oriented Models," *Proc. IEEE Real-Time Systems Symposium*, Dec. 1998, pp. 92-103.

[140] L. Sha, R. Rajkumar,and J.P. Lehoczky, "Priority-Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions On Computers*, Vol. 39, No. 9, Sept. 1990, pp. 1175-1185.

[141] L. Sha and J. Goodenogh, "Real Time Scheduling Theory and ADA," *IEEE Computer* , Vol. 23, Apr. 1990, pp. 53-62.

[142] O. Sokolsky, I. Lee, and H.Ben-Abdallah, "Specification and Analysis of Real-Time Systems with PARAGON," *Annals of Software Engineering*, 1999.

[143] J. Springintveld, F. Vaandrager, P. D'Argenio , "Testing Timed Automata ," To appear in *Theoretical Computer Science*, 2000.

[144] B. Sprunt, l. Sha, and J.P. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems," *Journal of Real Time Systems*, 1989, pp. 27-60.

[145] S. Tasiran, S. P. Khatri, S. Yovine, R. K. Brayton, and A. Sangiovanni-Vincentelli. "A Timed Automaton-Based Method for Accurate Computation of Circuit Delay in the Presence of Cross-Talk," *FMCAD'98*, 1998.

[146] S. Tripakis, and C. Courcobetis, "Extending PROMELA and SPIN for Real Time," *Proc. of the 2nd Int'l. Workshop Algorithms for the Construction and of Systems (TACAS'96)*, Lecture Notes in Computer Science 1055, Springer Verlag, 1996, pp. 329-348.

[147] S. Tripakis, "Timed Diagnostics for Reachability Properties," *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, TACAS'99*, Lecture Notes in Computer Science 1579, Springer-Verlag, 1999.

[148] S. Tripakis, S. Yovine, "Verification of the Fast Reservation Protocol with Delayed Transmission using the tool Kronos," *Proc. of the 4th IEEE Real-Time Technology and Applications Symposium, RTAS'98*, June 1998, IEEE Computer Society Press.

[149] S. Tripakis, and S. Yovine, "Analysis of Timed Systems based on Time-Abstracting Bisimulations," In *Proceedings of the 8th Inter. Conference On Computer Aided Verification, CAV'96*, Lecture Notes in Computer Science 1102, Springer-Verlag, July 1996. To appear in *Formal Methods in System Design*, Kluwer Academic Publisher, 2000.

[150] F. Vaandrager, N. Lynch, "Action Transducers and Timed Automata," *Proc. CONCUR'92*, LNCS, Vol 630, pp. 436-455, 1992.

[151] J. Verhoosel, D. Hammer, and E. Luit "A Model for Scheduling of Object-Based Distributed Real-Time Systems," *Real Time Systems Journal*, Kluwer Academic Publishers, Vol 8, pp. 5-34, 1995.

[152] S. Vestal, "Modeling and Verification of Real-Time Software using Extended Linear Hybrid Automata," *5th NSAS Langley Formal Methods Workshop*, 2000.

[153] J. Xu and D. Parnas "On Satisfying Timing Constraints in Hard-Real Time Systems," *IEEE Trans. On Software Eng.*, Vol. 19, No. 1, January 1993.

[154] Li Xuan Dong and Dang Van Hung, "Checking Linear Duration Invariants by Linear Programming," In Joxan Jaffar and Roland H. C. Yap, editors, *Concurrency and Paralellism, Programming, Networking, and Securiry*, Lecture Notes on Computer Science, vol. 1179 , pages 321–332, December 1996.

[155] Wang Yi, "Real-Time Behavior of Asynchronous Agents," *Proc. CONCUR'90*, Lecture Notes in Computer Science 458, Springer Verlag, 1990.

[156] Jin Yang, Aloysius K. Mok and Farn Wang, "Symbolic Model Checking for Event-Driven Real-time Systems," *ACM Trans. on Programming Languages and Systems*, March 1997.

[157] S. Yovine, *"Méthodes et Outiles pour la Vérification Symbolique de Systemès Temporisés,"* doctoral disertation, Insitut National Polytechnique de Grenoble, 1993.

[158] S. Yovine, "Model-Checking Timed Automata," *Embedded Systems,* G. Rozemberg and F. Vaandrager eds., Lecture Notes in Computer Science, Springer Verlag, Vol. 1494, October 1998.

[159] Zhou Chaochen, M.Hansen , A. Ravn, and H.Rishel, "Duration Specifications for Shared Processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no 2, pp. 170-183, 1990. Formal techniques in Real-Time and fault Tolerant Systems, vol. 571, Lecture Notes in Computer Science, Springer verlag, 1992.

[160] Zhou Chaochen, C. Hoare, and A. Ravn, "A Calculus of Durations," *Information Processing letters*, vol. 40, no. 5, pp:269–276, December 1991.

[161] Zhou Chaochen, Zhang Jingzhong, Yang Lu, and Li Xiaoshan, "Linear Duration Invariants," *Proc. Formal Techniques in Real-Time and Fault-Tolerant systems*, Lecture Notes in Computer Science, vol. 863 , 1994.

[162] K.Čerans, "Decidability of Bisimulation equivalence for Parallel Timer Processes," *Proc. of 4th Workshop on Computer Aided Verification*, Lecture Notes in Computer Science, 1992.

Appendix A

A Survey on RTS Design Notations and Analysis Tools

In this annex we present a classification and survey on design notations and models that are aimed at representing and reasoning on physical design for hard real-time systems. Then, we conclude some common characteristics and trade offs. Those remarks motivated our proposal that integrates techniques from two different fields (scheduling theory and state space exploration based methods) to cope with verification in usual application models.

A.1 Notations and Models for Physical Designs of RTS

We believe that in order to classify a technique as appropriate for physical design it should be able to deal with the temporal phenomena that arise from the processor sharing.
These aspects are usually ignored in the requirement/specification phases where timing information is expressed without taking into consideration how the described behavior will be implemented [33]. However, it is worth mentioning that some authors consider that the designer should only be responsible for conveying the global real-time requirements to the real-time scheduler (e.g.[77, 78, 79, 105, 151, 153],etc). These are "correct by construction" or "synthesis approaches" (which are, in general, based on pre-run time scheduling). Although we believe that those approaches may be ideal for the development of real-time software, as far as we know, they are still limited in practice.[1] Then, since we decided to address with analysis methods, we left out the scope of this survey all these synthesis methods.

[1]This might be due several different reasons: they might aim at particular time-driven applications, they might need not common OS platforms, the process of finding feasible solutions might posses high complexity, etc

A.1.1 A Tool Classification

There are many ways for classifying the approaches. As we are mainly concerned with correctness of designs, we choose to divide them according to the type and scope of the analysis which provide or suggest.

Based on Scheduling Results. This group is constituted by formal and informal notations that use the scheduling results for a periodic task model to asses timing requirements [50, 109, 12, 43]. These are usually deadline satisfaction requirements which come from sampling rate or response time requirements. No functionality is taken into account for analysis. The control structure of the task might be represented as the abstract response structure of the event to be served (sequential, parallel or select ordering of actions). Each action is described in terms of the way it consumes resources (amount, preemtiveness, priority, etc.) (e.g., [109]). All tasks are considered periodic and interruptions are handled using some kind of sporadic serving scheme ([144]). They also support certain degree of analysis in certain multiprocessing environment model [134, 24]. Their major strength is that they are of polynomial complexity on the number of tasks [86]. We can group these tools according to which kind of results are used.

- Based on Analytical Results. In this subgroup we find tools that use or suggest known results to statically asses the schedulability of a set of tasks. Examples are tools with a scheduling analyzer ([122, 13, 98, 97]). We also include methods based on semiformal design notations that suggest to apply analytical methods to carry on a timing correctness verification [42, 139, 82, 66].
- Based on Bounded Execution: Other tools use periodical and critical instant results for deriving schedulability property through a bounded simulation [93].

Based On State Space Exploration. This group is constituted by tools which use operational formal kernels whose verification techniques are based on state space exploration (reachability analysis, model checking). In general, they are able to solve a broader family of queries than the former group. Also their hypothesis on the application model are less restrictive than in the former approach (i.e., they support richer process models than those considered by scheduling theory, complex environment could be modeled, etc.). Their major drawback is the fact that they suffer from state space explosion problem and hence their associated verification problem has high time/space complexity if decidable at all.
Many operational formal methods and techniques were adapted to tackle difficulties that arise on physical design modeling. Process algebraic approaches are the most numerous [73, 57, 132, 31] but there are also Petri Net [71] and automata [58] based approaches. Another way to classify these tools is through their ability to model the preemption of operations. On one hand, there are tools able to model directly the effect of processor sharing on execution time of actions. They can be based on dense time domain [71, 58] or discrete time domain [46, 73]. On the other hand, there are tools which assumes that

actions execute atomically for an amount of time specified by the designer (preemptive disciplines can be somehow modeled by breaking a task into non-preemptive units). They can be based on a dense time domain [32, 70] or discrete time domain [132, 21].

Based On Deductive Reasoning. These approaches are based on semiautomated syntactical reasoning in a refinement or verification fashion [theorem provers, transformational tools, etc.]. By their own nature, deductive frameworks are much more prone to allow some integration with external knowledge (e.g. scheduling theory). For example, in [72] scheduling results are used to verify the feasibility of the transformations over a data flow net. In [119] and [120] a TLA approach is presented claiming for a separation of concerns between scheduler and the program correctness (abstracting away the scheduler as a scheduler policy when proving program correctness). Duration Calculus [160] approaches suit very well the specification of execution environment since duration is a particularly adequate concept to describe the preemptive nature of some scheduling policies [159, 49].

Based on Integer Programming. This group is the less numerous one. The key idea is to apply integer programming for synthesizing distance between events [16, 14]. Thus they proposed an alternative to space exploration techniques to solve some questions about temporal distance between events.

Note that, since we want to deal with hard real-time requirements stochastic techniques for performance evaluation were left out of the scope on purpose.

A.1.2 Guiding Features and Characteristics

In this section, we explain the different aspects that were studied in each different proposal. As they are of different nature (notation/analysis models, formal/informal notations) some of the aspects might not be applied in particular cases. Anyway, our goal is to keep the description as homogeneous as possible to allow an easy comparison.
In particular, the survey refers to the following features:

Modeling Features . This is a brief review of the notational characteristics of each approach. We try to point out some limitations on provided features that are circumstantial and do not respond to expressive constraints of the underlying formalism.

- **Components and Connectors.** Notations studied share some basic notion of concurrent sequential activities interacting and competing for resources to achieve systems goals. These are statically represented in the notation by some kind of software component. These components can be merely uncohesive tasks or the set of services provided by a class of objects. We describe the capabilities of the approach to express the distribution of the activities on processing resources. We also point

out how many sequential activities of a component could coexist in a system run. As notations should address the communication among sequential activities, we briefly describe the basic connectors provided by them.

- **Functional and Control Description of Components.** Some notations require / allow some type of description of the behavior of these activities at a control and/or functional level. Informal notations may merely suggest pseudocode and timing information while formal ones may require precise functional descriptions. We also try to point out whether special control flow schemes (e.g. exception handling) are supported.[2]

- **Time Issues.** Another point in common is that notations require/allow the designer to establish some timing information on software and external components that can be eventually used for the analysis. Indeed they serve as some sort of timing information repositories (local and global requirements like response times, sampling times, frequency of periodic outputs , estimations, etc.) [82, 87, 77]. We also point out the underlying time domain model (dense or discrete).

- **Processing Issues.** We analyze the way that actions are supposed to evolve in time (i.e. which is the granularity of actions, are they monolithic or is preemption modeled?). In particular we focus on the limitations to model operative platform characteristics like the scheduling policies supported or the run time overhead.

- **Environment Behavior Description.** Environment description plays an important role in the development of control systems. Notations address at some degree the description of the assumptions about the system to be controlled (also known as the plant).

Verification.

- **Underlying Formalism** [optional]. If the notation semantics is formally supported we report the class of formalism used.

- **Processor Sharing Modeling** [optional]. The formalisms that are used in this physical phase usually do not adhere the maximal parallelism assumption [73] which is a common feature of formalism aimed at specification phase where requirements are stated. One of the key aspects of a formal approach that supports preemption modeling is how the dichotomy between elapsed real-time and remaining processing time is handled. That is, we try to understand how maximal parallelism is abolished and how scheduling policy is modeled (i.e., it is built in, any one can be described, etc.).

- **Verification Technique.** To understand the verification approach we report the sort of technique applied (e.g. analytical worst case response time calculation, model checking or reachability analysis, etc.).

[2] We also describe any special capabilities to define connectors or hardware components.

- **Queries.** To understand the scope of the approach it is necessary to know what kind of properties can be queried (e.g. the ones provided by a logic or a fragment, ad hoc queries about minimal and maximal separations between events, etc.). It is also pointed out how they are formulated (logic, or manual adaptation to a reachability problem).

- **Treatment of Functional Aspects** [optional]. We believe that it is important to know to which extent data and functional aspects are taken into account in the verification phase (e.g. can queries involve data?, are infinite domains for data allowed?, etc.).

- **Complexity.** The modeling and the querying capabilities impact on the complexity to solve the general verification problem in the approach (is it computable, tractable). Some reported observations are presented.

Although approaches have different characteristics, goals and scopes they can be analyzed following the guide formerly presented. We think they could help to understanding the flavor, the utility and the limitations of approaches.

A.1.3 The Survey

The following detailed survey is not extensive but it describes some of the most representative informal notations and automatic techniques on this field. Since were interested in automatic (or potentially automatic) techniques deductive formal methods are not treated in further detail.

Approaches Based on Scheduling Results

- **SAD an AAD of ADARTS/CODARTS** [82]

  SAD (Software Architecture Diagram Notation) is a notation to describe systems through their task structure. AAD [82](Ada Architecture Diagram Notation) introduces ADA specific tasking notation thus providing ADA direct features like packages and rendezvous.

  - **Modeling Features.**
    * **Components and Connectors.** Task structures following the periodic task model [109] can be described at each node of a statically distributed system. At any instant, there is at most one executing instance of each task (1 job per task).
      The notation presents various mechanisms for loosely (prioritized or no prioritized) and tightly coupled message communication. Three types of event synchronization are possible: external, timer, and internal. An external

event is typically an interruption from external I/O devices. Information hiding modules are used for encapsulating data stores. Tasks access the data store indirectly via operations which manipulate the contents of data stores.

  * **Functional and control Description.** Although pseudocode serves as tasks behavior specification language, functional descriptions are ignored in the verification phase.
  * **Time Issues.** The Notation suggests the designer to define estimations of computation requirements, periodicity, deadlines, interarrival time for interruptions, etc.
  * **Processing Issues.** Each node is scheduled using a fixed priority policy. Its operative hypothesis could be as sophisticated as allowed by Rate Monotonic Analysis [109].
  * **Environment Behavior Description.** This tool is aimed at studying basically software artifacts behavior. The notation support the notion of event that can be externally generated. Timing information about the interarrival time of these external interruptions are used to perform the analysis of aperiodic tasks that serve them. Dataflows of passive read/write information could also be depicted.

- **Verification.**
  * **Verification Technique.** Rate Monotonic Analysis [109] is suggested to be applied using the annotated estimations.
  * **Queries.** Local and end-to-end deadlines satisfaction, jitter requirements, average response time for sporadic events.
  * **Complexity.** The formulas are of polynomial complexity respect to the number of tasks.

- **HRT-HOOD.** [42]

  HRT-HOOD is an object oriented notation that guides the designer to produce designs of high degree of analyzability in terms of scheduling theory (e.g. worst case blocking should be predictable). Another interesting semantical adaptation of an object oriented notation (ROOM) to deal with Hard Real-Time Systems development can be found in [75, 139].

  - **Modeling Features**
    * **Components and Connectors.** Objects are the components shown in the static representation of the system. The notation provides object "classes" common in hard real-time active abstractions: cyclic objects (for periodic activities) and sporadic objects. Static distributive computing is supported. The activities communicate trough requests (asynchronous - Loosely Coupled : blocked until ready to serve, and highly synchronous execution request: blocked until serviced, Rendezvous like) that might have an associated time out.

Passive (an information hiding version of shared memory) and protected objects (similar to Djykstra's Monitors) may act as data repositories.

* **Functional and control Description.** There is no functional or control description before the translation to ADA. Just the resource usage is described.
* **Time Issues.** Timing attributes should be associated with objects: cyclic (period, deadlines, offsets), sporadic (minimum arrival interval, offsets, deadlines). Depending on the type of the scheduling approach other attributes might be annotated.
* **Processing Issues.** Each node is scheduled using a fixed priority scheduling policy adhering to the models supported by the results of [39].
* **Environment Behavior Description.** This tool is aimed at studying basically software artifacts behavior. The notation supports the notion of event that can be externally generated. Timing information about the inter-arrival time of these external interruptions are used to perform the analysis of aperiodic tasks that attend them. Dataflows of passive read/write information could also depicted.

– **Verification.**

* **Verification Technique.** Schedulability analysis could be performed using the corresponding analytical theory (Worst case estimation using fixed priority scheduling theory [39]).
* **Queries.** Local and end-to-end deadlines satisfaction, jitter requirements, average response time for sporadic events.
* **Complexity.** The formulas are of polynomial complexity on the number of tasks.

• **PERTS** [122]

PERTS (A Prototyping Environment for Real Time Systems) its a prototyping environment for the evaluation of real-time systems. A key component is the schedulability analyzer. The basic version supports the analysis and validation of real-time systems built on framework of periodic task model. PERTS was chose as the example of analysis tool based on analytical results. A similar analysis capability is provided by STRESS [13]; in this case scheduling analyzer that provides is based on the results of [121, 118, 11] or commercial tools like iRAT [98] based on Rate Monotonic Analysis [109].

– **Modeling Features** Although a more complex task and resource model is available for the simulation environment it is not taken into account in the following description.

* **Components and Connectors.** Tasks and resources can be represented into the model using the periodic task model assumptions. Aperiodic tasks can be scheduled according to a variety of schemes including pure or persistent polling, deferrable server and sporadic server [144]. At any instant

there is at most one executing instance of each task (1 job per task). Tasks are statically assigned to processors.
Shared resources controlled by priority ceiling protocols, non-preemptive critical approach, priority inheritance or stack based protocol [140, 18].

* **Functional and Control Description.** There is no functional or control description. Just the resource use is described.
* **Timing Aspects.** Tasks are annotated with the worst case execution time, their deadline and their release time.
* **Processing Issues.** Processing rate of processors can be specified. Resources can be preemptable or not preemptable. In each node can be scheduled using a priority Driven scheduling (RM, Deadline Monotonic, EDF).
* **Environment Behavior Description.** This tool is aimed at studying basically software artifacts behavior, thus there is a limited support to environment modeling. External bounded or bursty events are treated as they were periodic.

– **Verification.**

* **Verification Technique** Analytical results for periodic task model are used [121, 134, 144, 118, 140, 18, 51]. The analysis of multiprocessor systems is based on the multiprocessor model [134] and end-to-end scheduling model [24].
* **Queries.** Deadline satisfaction, worst case execution time, blocking time, Jitter satisfaction, End to End analysis.
* **Complexity.** For monoprocessor systems, the formulas are of polynomial complexity on the number of tasks.

Approaches Based On State Space Analysis

• **RTD** [71]

It is a formal notation based on SA/RT requirement notation. Its semantics is given using High Level Timed Petri Nets [80] through graph grammar rules [19].

– **Modeling Features**

* **Components and Connectors.** It allows to describe the system as a set of tasks and terminators. At any instant there is at most one executing instance of each task (1 job per task). The version studied models monoprocessor systems. [3]
Connectors are inspired by POSIX [133] standards: message queues, shared memories, signals (user/O.S).
* **Functional and control description.** Each task comprises one or more sequential activities, the default thread and the exception threads (waken up by signals), that are executed in mutual exclusion. The internals of a task

[3] There is a tool for graphical language definition and RTD is just a particular instance of it [19].

are specified by means of data/control flow description. Functionality of the "bubbles" (processes) is expressed in an operational fragment of VDM. A Control specification facility is also available for controlling the activation of bubbles and describing the manipulation of signals.

* **Timing Aspects.** An expression, that could be data dependent, defines the upper and the lower bound of functional activities (bubbles). The period for cyclic tasks is another timing parameter of the notation. It is a dense-time model.
* **Processing Issues.** Actions are preemptable. The version studied allows fixed scheduling policy. Run time overhead is potentially modelable.
* **Environment Behavior Description.** Terminators are modeled as data streams independent from the received signals (however it could be given semantics in many details).

– **Verification.**

* **Underlying Formalism.** It is formally defined using HLTPN [80] as base formalism.
* **Processor Sharing Modeling.** In HLTPN the timestamps can be manipulated in such way that remaining execution time of an activity can be kept in a token. The scheduler is modeled as a subnet.
* **Verification Technique.** Simulation and symbolic execution is available. Also bounded reachability analysis and partial model checking.
* **Queries.** A C based language is available for bounded time queries [20].
* **Treatment of Functional Aspects.** Functional aspects are annotated in a fragment of VDM and translated as the functions to be performed by transition firings on the resulting tokens. Queries can predicate on data values.
* **Complexity.** The general reachability and model-checking problem are not decidebla on HLTPN.

• **The approach of CORBETT** [58]

It is an approach aimed at analyzing ADA code based on Hybrid Automata. However, it can also be seen as a technique to analyze ADA-like design notation. An extension to this work was presented to deals with partially implemented systems where queries are made in Graphic Interval Logic [15]. Another related approach can be found in [38] where parametric Hybrid Automata are used for synthesizing processor speeds or the amount of processor time given to each task to accomplish a performance goal.

– **Modeling features**

* **Components and Connectors.** Components are ADA tasks. At any instant there is at most one executing instance of each task (1 job per task). No dynamic task creation is modeled. This version deals with monoprocessor environments.

It supports Rendezvous communication and protected objects (similar to monitor) as communication media. It also models asynchronous transfer control which allows the completion of a blocking operation to abort the execution of a sequence of statements.

* **Functional and control Description**. Code written in an ADA fragment.
* **Time Issues**. Timing information is expressed by means of delay until operation and the estimations for bounding sequential code execution time. Continuous time make the time unit definition largely irrelevant.
* **Processing Issues**. Actions are preemptable. The major advantage of this modeling technique is that potentially allows reasoning in scheduling context where non scheduling theory is known or where the system does not fit any theory hypothesis like periodicity. Currently, it supports static priority scheduling accounting for run time overhead (task synchronization, timer services, context switches and interrupts).
* **Environment Behavior Description**. Separation between interruptions can be modeled to communicate the task system the occurrence of an external event.

– **Verification**.

* **Underlying Formalism** This approach presents a way to model ADA code into a Slope Linear Hybrid Automata [5] which allows the preemption modeling. They use automata nodes to represent the different states that the application and O.S. could reach. As they work with continuous time domains they get conservative abstractions of the software. They use virtual coarsening techniques [10] to reduce the size of the automaton.
* **Processor Sharing Modeling**. The scheduler is modeled into the state transition representation. The Hybrid automata allows to express the derivative of clocks. Then, derivative equal to one models that the task is running while derivative equal to zero models that the task is waiting for the processor.
* **Verification Technique**. They apply a symbolic model-checking tool for analyzing Hybrid Systems (HyTECH) [92].
* **Queries**. A property automaton is build to compose with the system and then reachability analysis is applied.
* **Treatment of Functional Aspects**. Some finite domain data modeling is allowed.
* **Complexity**. Unfortunately, the general model checking and reachability problem in Hybrid Automata is undecidable (they claim that most of the queries do terminate). State space explosion problem currently limits it to small size system. They believe that since task scheduling is deterministic, most nondeterminism in the transition system is caused by the timer transition, delay statements/alternatives and unmodeled program variables.

- **RTSL** [73]

  RTSL is an approach mathematically founded in process algebra. RTSL is used to describe the control/communication behavior, timing behavior, and the timing constraints (deadlines) of a monoprocessor system. The algebra provides high level real-time constructs like watchdogs and exception handlers.

  - **Modeling Features**
    * **Components and Connectors.** The system is modeled as a set sequential processes using a process description language whose syntax and semantics are based loosely on those of CCS [126] and ACP [17]. It models monoprocessor platforms.
      Complementary synchronization actions models synchronization channels (as Ada Select statement without data passing).
    * **Functional and control Description.** Process are built from actions. No data is modeled. This process algebraic approach is aimed at representing communicating structure of processes but taking into account duration of internal and communication actions as well. Timing exception handling can be modeled.
    * **Time Issues.** Internal and communication actions are labeled with their durations measured in time units. There are operators to express delay and deadline timing constraints. They use a discrete time model.
    * **Processing Issues.** It is a tick resolution approach that allows preemption modeling. The scheduling discipline is a modular component of the formal model so it can be changed. A parametric priority function arbiters the selection of the process to make it progress (RM, EDF can be modeled).
    * **Environment behavior description.** The environment can be modeled as processes which interact with the control software. All the language constructs are available for its description.
  - **Verification**
    * **Underlying Formalism.** RTSL is a process algebra with formal semantics.
    * **Processor Sharing Modeling.** The system semantics establishes that just a highest priority process advances while the others idle. The difference between elapsed time and executed time is reflected in time constructs where relative deadlines are reduced when the enclosed process idles.
    * **Verification Technique.** Reachability analysis is applied.
    * **Queries.** They present the deadline satisfaction problem as a reachability question since states record no-handled exceptions that arise when deadlines are missed.
    * **Complexity.** State space explosion problem may arise. However they hypothesize the filtering effect of scheduling disciplines and the regular structure of many real-time systems will tend to control that problem in practice.

- **VERUS** [46]

VERUS is a tool for quantitative analysis of finite state real-time systems based on Labeled Transition Systems.

- **Modeling Features**
  - * **Components and Connectors.** Task can be described in a C-like syntax. Each task corresponds to one executing sequential activity. It models monoprocessor platforms.
    Shared variables are available for task communication. The can be protected accessing at a high priority level. Other protocols or communication models can be implemented as well.
  - * **Functional and control Description.** It is done trough a C-like syntax. Data types allowed are integer and boolean. No interruption features can be directly modeled.
  - * **Time Issues.** The language provides primitives to express timing aspects such as deadlines, priorities, and time delays in an imperative fashion. VERUS language has a wait operator to model discrete time duration of actions.
  - * **Processing Issues** They adopt tick resolution interleaving. Like most timed formalism action visible state changes are instantaneous. The wait statement controls time elapse. The discipline modeled is fixed priority scheduling (subtasks can have different priorities).
  - * **Environment Behavior Description.** This tool is aimed at studying basically software artifacts behavior. Periodic task statement is used to describe indirectly actions performed by a sporadic task serving external request.
- **Verification.**
  - * **Underlying Formalism.** The language is translated to untimed transition systems with global variables.
  - * **Processor Sharing Modeling.** The interleaving semantics of transition Systems models the shared processor.
  - * **Verification Technique.** It is an adaptation of the symbolic model checking technique of [36]. The algorithms provide valuable timing information as minimum, maximum delay (counting the number of transitions), condition counting (max and min. number of times a given condition holds on any path from starting to final sate).
  - * **Queries.** An Extension of symbolic CTL (RTCTL) is used for expressing time bounded properties.
  - * **Complexity Issues.** Tick resolution interleaving might exacerbate the state space explosion phenomena [132].
  - * **Influence of Functional Aspects.** Finite data can be used since the language is translated to a labeled transition graph.

- **ARC** [132]

  It is formalism that extends CSP (Communicating Sequential Processes) [94] with a global clock.

  - **Modeling Features**
    - **Components and Connectors**. Components are the (CSP) processes (tasks). It models monoprocessor platforms.
      Basic connector is the synchronous Communication (CSP like). The two processes involved in the communication synchronize in an "out" and "in" operations.
    - **Functional and control Description**. The description is given by means of untimed CSP description showing the possible behavior in terms of the communication structure. No functional or data aspects are specified. No interruption features can be modeled.
    - **Time Issues**. W(n) events represents the advance of time in n units. The time model is discrete.
    - **Processing Issues**. The language only addresses non-preemptive fixed priority scheduling. Actions are monolithic.
    - **Environment behavior description**. The environment can be modeled as processes that interacts with the control software. All the language constructs are available for its description.
  - Verification
    - **Underlying Formalism**. Is an extension of untimed CSP. A Global, discrete clock is increased by time advancing W(n) events.
    - **Processor Sharing Modeling**. The interleaving semantics and the W(n) events models a shared processor environment.
    - **Verification Technique**. ARC is compiled into an Labeled Transition System. Then refinement and equivalence checking can be applied.
    - **Queries**. It can be asked if refinement or equivalence holds between terms modeling systems.
    - **Complexity**. It solves the extra production of states of tick resolution using action resolution. Anyway, as all state space exploration based approaches, it potentially suffers from state space explosion problem.

- **GCSR** [21]

  Graphical Communicating Shared Resources is a graphical language to describe RTS in a Structured-Charts fashion. GCSR supports the explicit representation of system resources and priorities to arbitrate resource contentions.

  - **Modeling Features**
    * **Components and Connectors.** Real-time systems are basically sets of communicating processes. It supports multiprocessor platform modeling. Processes can synchronize through matching event names. The interruption or communication can occur during execution of a process.
    * **Functional and control Description.** Control description is given trough a set of nodes (to model time-consuming actions) connected through directed edges (modeling events like: communication, time-out, etc.).
    * **Time Issues.** In discrete model the actions takes one tick and it can be established which resources they use and the priorities over them, events are timeless and they also have priorities assigned to them. It is possible to express the actions to be taken after a time-out an external or an internal interruption.
    * **Processing Issues.** It is aimed at modeling Fixed Priority scheduling. Preemption of actions can be modeled dividing the task into non-preemptable units [142].
    * **Environment behavior description.** The environment can be modeled as agents that interact with the control software. All the language constructs are available for its description.
  - **Verification.**
    * **Underlying Formalism.** The Graphical Language is translated into ACSR [117] a process algebraic approach that takes into account that resources are shared. Contention for resources is arbitrated according to fixed and static priorities.
    * **Processor Sharing Modeling.** Processors are resources. Actions that compete for them are arbitrated according a fixed priority scheme.
    * **Verification Technique.** There is a tool [57, 142] supporting the process algebraic view that provides bisimulation algorithms to compare designs against specifications. Reachability analysis tools are also provided.
    * **Queries.** Queries can be solved if they are rephrased as a reachability problem.
    * **Complexity.** As all state exploration based approaches it suffers from state space explosion problem.

Approaches Based On Linear Integer Programming.

- **The approach of AUVRUNIN et al.** [16]

  Avrunim et al. present a technique for deriving upper and lower bounds on the time that can elapse between two given events in an execution of a concurrent software system running on a single processor under arbitrary scheduling.

  - **Modeling Features**
    - **Components and Connectors.** A static set of processes. There is no possibility to model dynamic task creation. This version deals only with monoprocessors but it has been extended to treat distributed systems as well [14].
      Synchronous or asynchronous communication events among processes can be modeled.
    - **Functional and control Description.** Only the control structure of processes is described as a deterministic finite automata. There is no guide to model interruption features.
    - **Time Issues.** Each event must be assigned a duration. Upper bounds for cycles must be provided by the user. Time domain is irrelevant in this approach.
    - **Processing Issues.** Events could be preemptable activities that do not overlap. Actually, atomicity has no importance in this approach due to the technique used for calculating the distance between events. They consider complex software characteristics where system developer has little control over scheduling beyond that provided by the semantics of interprocess communication (incapable of verifying properties when the scheduling policy is essential to timing correctness).
    - **Environment Behavior Description.** This tool is aimed at studying basically software artifacts behavior. No external behavior can be directly modeled (It is aimed at analyzing a complex interaction of aperiodic processes in a transaction firing).
  - **Verification.**
    - **Underlying Formalism.** A system run is a coherent interleaving the events that compose the processes. These potential runs are constrained by a set of inequalities.
    - **Processor Sharing Modeling.** The objective function to be optimized is the sum of the durations of events (no overlapping). The range of this function depends on the desired time domain.
    - **Verification Technique.** They model the executions using linear inequalities and they use linear programming techniques instead of generating state spaces. The inequalities system are deduced to constitute the necessary conditions that must be satisfied by subsequences of events
    - **Queries.** Upper and lower bounds for a pair of events in a system run.

* **Complexity** It inherits the complexity of mixed integer linear programming problem however some good results are reported.
* **Treatment of Functional Aspects.** Only finite state processes can be modeled.

A.1.4 Some Remarks

Informal notations are richer in high level features providing languages closer to user needs. Obviously, due to its informal nature there is no automatic support for verification phase and this procedure might be error-prone. Tools like scheduling analyzers solve this fact but they still just asses the deadline meeting of the set of task. On the other hand, in most formal languages it is difficult to describe the system in terms of the available mechanisms for its eventual implementation. A dual notation approach (i.e., user-friendly notation translated into a formal kernel) is a paramount issue [19]. Unfortunately, the expressive power of formal languages necessary to feature real phenomena impact the feasability of verification [58, 71].

Functional aspects and data treatment are neither modeled nor taken into account in verification process of most approaches. Timing properties are analyzed independently from functional characteristics. If "separation of concerns" were the criterion followed by designers then temporal correctness should not depend on data itself but in the estimations of worst-case computation time. As a consequence, most methods do not directly address data modeling which is conservatively replaced by the introduction of some non-determinism in control and timing description.

Regarding the environment modeling, we can divide the approaches in two groups: the ones which can represent the interacting external agents in a rather complex patterns and the ones which indirectly address the issue through the possibility of modeling interruptions arrivals into the software system. Obviously, the former group usually allows a more sophisticated contextual analysis than the later where only interarrival parameters are given.

We observe that most formalisms has the scheduling policy (generally, fixed priority based) tacitly built in their semantics. The approaches that potentially provide more flexibility model the scheduler just as another component but with the drawback of generating some twisted and heavy dependence among components and the scheduler (e.g. [58, 71]). A third group is constituted by formalisms where the scheduling has its own syntactical and semantical category providing certain flexibility and encouraging a smoother transition from requirements [73].

We can also conclude that one of the main obstacles in the efficiency of property verification is the granularity of actions. While in discrete fully preemptive models the tick resolution could produce unnecessary states [132], on the other hand, dense time

preemptive models like Hybrid-Automata or HLTPN has undecidable verification problems [5] or its complexity could be prohibitive. Other well known specification formalisms like Temporal Automata [7, 130, 103] based approaches, most process algebra and Petri Nets approaches are not able to model preemption. They are oriented to a maximal parallelism assumption that is the right approach for a specification. For example in the case of automata, time clocks can be used to measure the minimum and maximum elapsed time in a node. Unfortunately, as clocks can not be "stopped" as in a Hybrid-automata, it is not possible to accurately model preemption.

In the formal approaches, the RTS models are either monolithic or they are composed by a set of terms which behaviors are heavily interdependent since they share the same processing resource. At some extent, it is true that, as some authors says, scheduling policy reduces the number of behaviors of the system. However, if n is the number of tasks, in a system the possible configurations of a Ready queue is 2^n, the number of subsets (imagine an event driven system) . We can guess that the states in which the preempted tasks were frozen are other possible sources of combinatorial explosion. That is, scheduling reduces the number of behaviors wrt. a system where an arbitrary scheduler is assumed. Nevertheless, the number of tasks still has a negative impact on the practicality [58].

Our last observation, that originally inspired our work, is that state space exploration methods are usually motivated by the need of reasoning on non standard situations where there is no known scheduling theory. Nevertheless, when scheduling theories are known they are not exploited in state based verification techniques.

Appendix B

Abstract Code for the Working Examples

The Active Structure Control Design

```
Modeler (Sporadic, MAT = 100, Pty= 10,
         Interruption=DataReceived, LatchUpTo 1 at WaitingForReplenish)

Init: [10]{ModelerReadyForComm.}
Begin
[18,23]; Model.UpDate; [10] {ModelerReadyForComm.}
End


Pulser (Periodic, P = 110, Pty= 11)
Begin
[1];Model.Read;[38,43];[10]{PulserReadyForComm.}
End


Model (Protected, PtyCeiling = 12)
Begin
    Operation: UpDate
    Begin
    [2]{ModelUpDated.}
    End

    Operation: Read
    Begin
    [1]{MoldelRead.}
    End
End
```

The Mine-Pump Design

This is the code of the Mine-Pump Design.

Local Processor runs :

```
HLW-Sensor (Sporadic, MAT = 6000 ms, Pty= 5,
            Interruption=ChangeReceived, LatchUpTo 1 at Always, IH=1 ms,
           PtyIH=12)
Begin
[1]<dataread>;Motor.SetPump;Logger.Log;[1]
End

CH4-Sensor (Cyclic, Period = 80 ms, Pty= 10)
Begin
[2]; ({CH4sensorNotReady.}
        Console.Alarm
     +
      {CH4done.}
        [.5]{CH4read};[1];CH4Status.Read;[1];
         ({NotSafe.}
           Motor.NotSafe;CH4Status.Write;
           Console.Alarm
         +
          {Safe.}
           Motor.Safe;CH4Status.Write
          );Logger.Log
      );
[.5]{CH4set.}
End

AirFlow-Sensor (Cyclic, Period = 100 ms, Pty= 7)
Begin
[.5]{flowread.};[1];
({nok} Console.Alarm + {ok}); Logger.Log;[.5]
End

CO-Sensor (Cyclic, Period = 100 ms, Pty= 9)
Begin
[2];({COdone.}
       [.5]{coread.};[1];
       ({nok} Console.Alarm + {ok});
     +
     {COnotready}
       Console.Alarm
     );Logger.Log;
[.5]{COset.}
End

WaterFlow-Sensor (Cyclic, Period = 1000 ms,
                  Pty= 8)
Begin
[.5]; Motor.RequestStatus;[.5]{wflowread.};
```

```
[5];({nok} Console.Alarm + {ok});
[1];Logger.Log;[.5]
End

HLW-WatchDog (Cyclic, Period = 500 ms, Pty= 6)
Begin
[1];AckQ.Extract({AckQExtracted_{OK}.}
                          [1]{ackrequested}
                +
                 {AckQExtracted_{NOK}.}
                          Console.Alarm
                 );[.5]
End


ACK-Handler(Sporadic, MAT = 500 ms,
LatchUpto 1 at WaitForReplenish,Pty= 12)

Begin
[1];AckQ.Add;[.5]
Puro IH: Evento AckReceived.
End

Console-Proxy (Cyclic, Period = 300 ms, Pty= 12)
Begin
[.5];Console.GetPckg;ConsoleNet.Send;[.5]
End

Command (Sporadic, Period = 1000 ms, LatchUpTo 0, Pty= 4)
Begin
[1];(Motor.RequestStatus + Motor.SetPump);[1];
({statusrequired} Console.InfStatus
+{notrequired})
End

AckQ (Protected, PtyCeiling = 13)
Begin
    Operation: Add
    Begin
    [2]{AckQAdded_OK, AckQAdded_NOK}
    End

    Operation: Extracted
    Begin
    [2]{AckQExtracted_OK, AckQExtracted_NOK}
    End
End

ConsoleNet (Protected, PtyCeiling = 12)
Begin
    Operation: Send
    Begin
    [1]{CNetSent_OK, CNetSent_NOK}
    End
```

```
End

Console (Protected, PtyCeiling = 13)
Begin
    Operation: Alarm
    Begin
    [2]{alarmadded}
    End

    Operation: GetPckg
    Begin
    [1]{pckgget}
    End

    Operation: InfStatus
    Begin
    [1]{infStatusadded}
    End
End

Logger (Protected, PtyCeiling = 13)
Begin
    Operation: Log
    Begin
    [2]
    End
End

CH4Status (Protected, PtyCeiling = 12)
Begin
    Operation: Read
    Begin
    [.5]{statusread}
    End

    Operation: Write
    Begin
    [.5]{statuswritten}
    End
End

Motor (Protected, PtyCeiling = 11)
Begin
    Operation: NotSafe
    Begin
    [1];({motoron} [.5]{motorclear}; Logger.Log +
         {motoroff});[.5];Logger.Log;[.5]
    End

    Operation: Safe
    Begin
    [1]; ({motoron} [.5]{motorset}; Logger.Log
         + {motoroff}
         );
```

```
    [.5];Logger.Log;[.5]
    End

    Operation: RequestStatus
    Begin
    [1]
    End

    Operation: SetPump
    Begin
    [.5];({toset}[.5];
                ({exceptionarisen}
                +
                 {ok}CH4.StatusRead;[.5];
                   ({exceptionarisen}
                   +
                    {ok} [.5]{motorset};
                         Logger.Log
                   )
                )
          +
           {toclear}[.5];({}[.5]{motorclear};
                                Logger.Log
                         +{}
                         )
          );
    [.5]
    End
End
```