

## DECLARATION OF WELMOED SPAHR

I, Welmoed Spahr, am over twenty-one (21) years of age. I have never been convicted of a felony, and I am fully competent to make this declaration. I declare the following to be true to the best of my knowledge, information and belief:

1. I am Vice President, Applied Computing Books, employed by Springer Science + Business Media LLC in New York (“**Springer**”).
2. Springer is a neutral third party in this dispute. I am not being compensated for this declaration.
3. Among my responsibilities as Vice President, Applied Computing Books, I manage an international group of employees that handle our Computer Science Books portfolio. This portfolio includes a series called “Lecture Notes in Computer Science.”
4. I make this declaration based on my experience in academic publishing, and my examination of the MARC information contained in Exhibits C and D.
5. As part of its ordinary course of business, Springer publishes and makes available to subscribers a series of books called “Lecture Notes in Computer Science.” These books contain conference papers that were presented to the public at various academic conferences.
6. It is the regular practice of Springer to publish the “Lecture Notes in Computer Science” book series and to make them available to the public in the form of printed books. As a matter of regular practice, Springer maintains copies and records of its publications in the ordinary course of its regularly conducted activities, including for the “Lecture Notes in Computer Science” book series.
7. The articles below have been attached as Exhibits A and B to this declaration:

|   |                                                                                                                                                                                                                                                                                                                                                          |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A | Yael Abarbanel, Ilan Beer, Leonid Gluhovsky, Sharon Keidar & Yaron Wolfsthal, "FoCs — Automatic Generation of Simulation Checkers from Formal Specifications", Lecture Notes in Computer Science 1855, 12 <sup>th</sup> International Conference on Computer Aided Verification 2000, Chicago, IL, USA, July 2000 Proceedings (hereinafter, “ABARBANEL”) |
| B | Konrad Feyerabend & Bernhard Josko, "A visual formalism for real time requirement specifications", Lecture Notes in Computer Science 1231, 4 <sup>th</sup> International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software, ARTS’97 Palma, Mallorca, Spain, May 1997 Proceedings (hereinafter, “FEYERABEND”)                   |

8. I obtained copies of Exhibits A and B from Springer's publication records, where copies are maintained in the ordinary course of Springer's business. Exhibit A is a true and correct copy of ABARBANEL as it existed in July 2000. Exhibit B is a true and correct copy of FEYERABEND, as it existed in May 1997.
9. Springer's publication records show the date of publication for these articles. I have attached Springer's publication records to this Declaration. The attached Exhibit C shows the publication date for ABARBANEL and Exhibit D shows the publication dates for FEYERABEND. MARC (machine-readable cataloging) is a standard set of digital formats for the machine-readable description of items cataloged by libraries.
10. As indicated by Springer's publication records, ABARBANEL was published in the Lecture Notes in Computer Science 1855, 12th International Conference on Computer Aided Verification 2000, Chicago, IL, USA, July 2000 Proceedings, which was published by Springer in July 2000. Copies of the conference proceedings, including ABARBANEL, would have been made available no later than the last day of the conference, which was July 15-19, 2000.
11. As indicated by Springer's publication records, FEYERABEND was published in the Lecture Notes in Computer Science 1231, 4th International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software, ARTS'97 Palma, Mallorca, Spain, May 1997 Proceedings, which was published by Springer in May 1997. Copies of the conference proceedings, including FEYERABEND, would have been made available no later than the last day of the conference, which was May 21-23, 1997.
12. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001.

I declare under penalty of perjury that the foregoing statements are true and correct.

Executed on: 19 December 2023

DocuSigned by:  
*Welmord Spahr*  
32AE3039DB354B0...

# **EXHIBIT A**

# Automatic Generation of Simulation Checkers from Formal Specifications

Yael Abarbanel, Ilan Beer, Leonid Gluhovsky,  
Sharon Keidar, and Yaron Wolfsthal

IBM Haifa Research Laboratory, Israel  
{yaell,beer,leonid,sharon,wolfstal}@il.ibm.com

## 1 Introduction and Motivation

For the foreseeable future, industrial hardware design will continue to use both simulation and model checking in the design verification process. To date, these techniques are applied in isolation using different tools and methodologies, and different formulations of the problem. This results in cumulative high cost and little (if any) cross-leverage of the individual advantages of simulation and formal verification.

With the goal of effectively and advantageously exploiting the co-existence of simulation and model checking, we have developed a tool called **FoCs** ("Formal Checkers"). FoCs, implemented as an independent component of the RuleBase toolset, takes RCTL<sup>1</sup> properties as input and translates them into VHDL programs ("checkers") which are integrated into the simulation environment and monitor simulation on a cycle-by-cycle basis for violations of the property.

Checkers, also called Functional Checkers, are not a new concept: manually-written checkers are a traditional part of simulation environments (cf. [GB+99]). Checkers facilitate massive random testing, because they automate test results analysis. Moreover, checkers facilitate the analysis of intermediate results, and therefore save debugging effort by identifying problems directly - "as they happen", and by pointing more accurately to the source of the problems.

However, the manual writing and maintenance of checkers is a notoriously high-cost and labor-intensive effort, especially if the properties to be verified are complex temporal ones. For instance, in the case of a checker for a design with overlapping transactions (explained in Section 3), writing a checker manually is an excruciating error-prone effort.

Observing the inefficient process of manual checker writing in ongoing IBM projects has inspired the development of FoCs, as a means for automatically generating checkers from formal specifications. For each property of the specification, represented as an RCTL formula, FoCs generates a checker for simulation. This checker essentially implements a state machine which will enter an error

<sup>1</sup> RCTL includes a rich and useful set of CTL safety formulas and regular expressions, see [BBL98]

formula fails to hold in this run. The next section  
 tion process in more detail.

Experience with FoCs in multiple projects has been very favorable in terms of verification cost and quality. Verification effort is reduced by leveraging the same formal rules for model checking of small design blocks as well as for simulation analysis across all higher simulation levels. An equally important benefit of FoCs is the conciseness and expressiveness of RCTL formulas. Formulas consisting of just a few lines can efficiently represent complex and subtle cases, which would require many lines of code if described in a language such as VHDL. This makes maintenance, debugging, porting and reuse of specifications and checkers highly cost-effective.

## 2 Tool Architecture and Implementation

Figure 1 shows the overall environment in which FoCs operates. The user provides a design to be verified, as well as formal specifications and a set of test programs generated either manually or automatically. FoCs translates the formal specification into checkers, which are then linked to the design and simulated with it. During simulation, the checker produces indications of property violations. It is up to the user to decide what action to take: fix the design, the property, or the simulation environment.

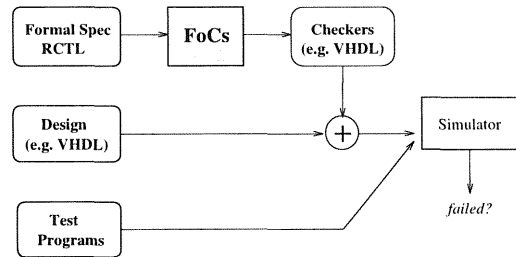


Fig. 1. FoCs Environment

FoCs translates RCTL into VHDL as follows: First, each property is translated into a non-deterministic finite automaton (NFA) and a simple  $AG(p)$  formula, where  $p$  is a Boolean expression. The NFA has a set of distinguished error states, and the formula specifies that the NFA will never enter an error state (entering an error state means that the design does not adhere to the specification under the test conditions). The translation details are described in [BBL98].

Since contemporary simulators do not support non-determinism, the NFA has to be converted into a deterministic automaton (DFA). The DFA, in turn, is translated into a VHDL process - the FoCs checker. The  $AG(p)$  formula is translated into a VHDL *Assert*( $p$ ) statement that prints a message when the VHDL process reaches a state where the underlying property is violated, and possibly stops simulation.

he DFA may be exponential in the number of  
ion is sensitive to the size of the representation  
(the number of VHDL lines) rather than to the number of states. The number  
of VHDL lines in the resulting VHDL checker is at most quadratic in the size  
of the property. Practically, it is almost always linear because of the types of  
properties that people tend to write.

The above translation process is implemented within the RuleBase model  
checker [BBEL96].

### 3 Example

The following example will demonstrate the conciseness and ease of use of RCTL  
for checker writing and the advantage of automatic checker generation. Assume  
that the following property is part of the specification: **If a transaction that  
starts with tag t has to send k bytes, then at the end of the transaction  
k bytes have been sent.** The user can formulate this property in RCTL as  
follows:

*forall k:* <sup>2</sup>

$\{*, start\_tag=t \ \& \ to\_send=k, !end*, end \ \& \ end\_tag=t\} \rightarrow \{sent=k\}$

Manual writing of a checker for this property may become complicated if transac-  
tions may overlap, which means that a new transaction may start while previous  
transactions are still active. The checker writer has to take into consideration  
all possible combinations of intervals and perform non-trivial bookkeeping. The  
RCTL formula is evidently much more concise and readable than the resulting  
VHDL file or a manually written VHDL or C program.

### 4 Using FoCs for Coverage Analysis

The quality of simulation-based verification depends not only on thorough check-  
ing of results, but also on the quality of the tests used (a.k.a. input patterns or  
test vectors) [KN96]. FoCs checkers can serve to enhance the quality of tests by  
providing a means for measuring *test coverage*. Test coverage measurement is the  
process of recording if certain user-defined events occurred during simulation of a  
set of tests. When used for coverage purposes, the FoCs checkers will evaluate the  
quality of the test suite by discovering events, or scenarios, that never happen  
during simulation. This feedback will guide the user which further tests are  
needed in order to cover scenarios that have not been exercised.

The implementation of coverage checkers is similar to that of functional  
checkers. The only difference is that instead of reporting an error, a coverage  
checker provides a positive indication when covering the relevant scenario. An

<sup>2</sup> The semantics of *forall* are intuitive. The implementation, however, is not trivial.  
It involves spawning a new automaton whenever a new value of k is encountered.  
Neither the formal semantics nor the implementation are included here due to lack  
of space.

is: **a snoop event happens twice between**  
 formulated in RCTL as follows:

$$\{*, read, \{!snoop*, snoop, !snoop*, snoop\} \mathcal{E} \{!write*\} \} \rightarrow \{cover\}$$

A special case of coverage analysis is detection of *simulation vacuity*. While vacuity in model checking is defined as a failure of a subformula to influence the model checking results [BBER97], simulation vacuity refers to the failure of a set of tests to trigger a functional checker, which means that the checker did not influence simulation results. To detect simulation vacuity, FoCs attaches a coverage checker to each functional checker it generates; the coverage checker indicates whether the functional checker was triggered in at least one simulation run.

## 5 Experience

The FoCs toolset has been deployed in several projects in IBM, notably in the GigaHertz processor development effort in Austin and in the IBM Haifa ASIC development laboratory. FoCs has also been successfully used by the formal verification team of Galileo Technology, Inc. The experience with FoCs in these projects has been very favorable in terms of verification cost and quality. Using FoCs, verification effort was reduced - reportedly by up to 50% - by using the same formal rules for model checking at the unit level and for simulation analysis in the subsystem and system levels. This reduction was achieved despite the fact that the addition of checkers increases simulation time considerably (up to a factor of two). Thousands of FoCs checkers were written so far by virtue of the great ease of writing RCTL specifications and translating them to FoCs checkers.

## 6 Related Work

In a previous work, Kaufmann et al [KMP98] described an approach to verification in which all specifications and assumptions have the form  $AG(p)$ , where  $p$  is a Boolean formula. Both specifications and assumptions of this form can be used in simulation, with specification being translated into simple checkers, and assumptions being translated into testbench code which avoids leading the simulation into undesired states.

Canfield et al [CES97] describe a platform called Sherlock, aiming to serve both model checking (through translation to CTL) and simulation. Sherlock includes a high level language for specifying reactive behavior, while we use RCTL - a regular expression based language. While Sherlock postprocesses simulation traces, our method works during simulation. No experience has been described in [CES97]. Our own experience has demonstrated that the simple, concise syntax and semantics of RCTL, coupled with online simulation checking, is highly useful to the verification teams with whom we work.

et al describe a methodology and tool that inspiral verification and simulation efforts by automatically translating state machines which represent the environment specification either to simulation behavioral models or to input for a model checker. Boolean assertions attached to the state machines serve as  $AG(p)$  formulas in model checking or  $Assert(p)$  in simulation. In contrast, our solution represents the specification as temporal logic formulas rather than state machines, for the sake of conciseness and readability.

## 7 Future Plans

Although focused on checker generation for functional testing and coverage analysis, we view FoCs as a step towards a full methodology of "Formal Specification, Design and Verification". We intend to provide a set of integrated, complementary tools that will facilitate the use of formal specification for multiple purposes. Once written, the formal specification will:

- Serve as an executable specification; architects can experiment with it while defining the specification
- Be used as a golden model to resolve ambiguities and misunderstandings during the implementation stage
- Be translated into temporal formulas for model checking
- Be translated into simulation checkers
- Be used for derivation of coverage criteria
- Provide hints for automatic test generation

We believe that such a methodology, once supported by the appropriate tools, will significantly contribute to the quality and efficiency of the design process.

## References

- [BBEL96] I. Beer, S. Ben-David, C. Eisner, A. Landver, "RuleBase: an Industry-Oriented Formal Verification Tool", Proc. DAC'96, pp. 655-660.
- [BBER97] I. Beer, S. Ben-David, C. Eisner, Y. Rodeh, "Efficient Detection of Vacuity in ACTL Formulas", CAV'97, LNCS 1254, pp. 279-290.
- [BBL98] I. Beer, S. Ben-David, A. Landver, "On-The-Fly Model Checking of RCTL Formulas", CAV'98, LNCS 1427, pp. 184-194.
- [CES97] W. Canfield, E.A. Emerson, A. Saha, "Checking Formal Specifications under Simulations" Proc. ICCD'97.
- [GB+99] D. Geist, G. Biran, T. Arons, Y. Nustov, M. Slavkin, M. Farkash, K. Holtz, A. Long, D. King, S. Barret, "A Methodology for Verification of a System on Chip", Proc. DAC'99.
- [KN96] M. Kantrowitz, L. M. Noack, "I'm Done Simulating; Now What? Verification Coverage Analysis and Correctness ", Proc. DAC'96.
- [KMP98] M. Kaufmann, A. Martin, C. Pixley, "Design Constraints in Symbolic Model Checking", CAV '98, LNCS 1427, pp. 477-487.
- [SB+97] T. Schlipf, T. Buechner, R. Fritz, M. Helms and J. Koehl, "Formal verification Made Easy", IBM Journal of R&D, Vol. 41, No. 4/5 , 1997.

# **EXHIBIT B**

# A Visual Formalism for Real Time Requirement Specifications

Konrad Feyerabend<sup>1\*</sup> and Bernhard Josko<sup>2\*\*</sup>

<sup>1</sup> Dept. of Computer Science, Oldenburg University, D-26111 Oldenburg, Germany

<sup>2</sup> OFFIS, Escherweg 2, D-26121 Oldenburg, Germany

**Abstract.** This paper presents a semantical basis of a graphical specification language, called *real-time symbolic timing diagrams (RTSTD)*, to express real-time requirements of embedded systems. RTSTD allow a concise and unambiguous formulation of real-time properties that are intuitively understandable by hardware designers. We give a precise semantical foundation of this graphical language in terms of real-time temporal logic. Due to this interpretation RTSTD can be embedded into existing verification tools to check whether an implementation satisfies the given specification expressed as RTSTD.

## 1 Introduction

Nowadays embedded control systems can be found in varying areas such as automotive, production technique, plant control and avionics. Essential for these control systems are their *reactive* nature. Furthermore, these systems are often safety-critical where an error may lead to catastrophic results for people, material asset, and environment. These safety requirements include real time constraints: the control system has to react to asynchronously occurring events within a given period to ensure a fault free behaviour of the system. For example, during a car crash the inflation of the airbag should neither be triggered too early nor too late to protect the driver. If the airbag does not work correctly the injury caused may be worse than without any safety device.

To increase reliability of safety-critical systems formal proofs that requirements are indeed satisfied are highly demanded. To prove properties of control-dominated reactive systems model checking techniques have been developed in the past. Since the introduction of symbolic, BDD-based techniques [5] these methods have been successfully applied in industrial designs. Using model checking, requirements are expressed in temporal logics. But temporal logics formulae are not well accepted by engineers who require a more readable specification language. To meet these needs a user-friendly notation called symbolic timing diagrams (STD) has been developed [6] that resemble the graphical appeal of conventional timing diagrams as widely used in hardware design. STD have a

---

\* Now at Ericsson Eurolab Deutschland GmbH, D-31135 Hildesheim, Germany

\*\* Part of this work has been funded by the Commission of the European Communities under the ESPRIT project 20897, SACRES.

well-defined semantics given in temporal logics and have been used successfully in several applications within the ESPRIT project FORMAT [14] by industrial partners.

To deal with real time constraints, timed automata and real time temporal logics have been investigated in recent years [2,1], and there are already quite a few model verification environments available (e.g. [7,15,3]) that can handle real time. But adding real time to temporal logics makes the specifications even more difficult to read and makes the need for an intuitive specification style even more pressing.

This paper deals with the extension of STD to real time. We give a formal semantics of these timing diagrams based on real-time temporal logics. While the extension of the graphical notation of STD to include timing constraints is quite obvious, its semantical interpretation is not. In the untimed version a requested event may be arbitrarily long delayed. Including timing information this is no longer possible. Due to this fact, we have to resolve possible conflicts of different types of constraints – whether a violation of a part of the constraints is interpreted as a failure or only as an exit of the diagram. The semantics given in this paper solves this conflict by introducing a priority on the constraint types. We will explain in more details and justify our decision in Sect. 5. Real time STD (RTSTD) will be used within an integrated tool suit for the specification and verification of VHDL [12] and StateMate [10] designs.

*Related Work.* Combining real time temporal logic with a graphical interface has been studied by Melliar-Smith et al. in [16] where the (purely graphical) real time future interval logic RTFIL is introduced. The graphical notation and the logics used there is quite different from our approach which makes comparison difficult. Dietz [8] uses duration calculus as semantic foundation of her graphical specification language. While duration calculus is extremely expressive it is shown to be undecidable and thus not suitable for automated proof.

Formalizing timing diagrams for interface specifications has been studied by Borriello ([4]). Khordoc and Cerny [13] study realizability of timing diagrams and their use in testing designs. To the best of our knowledge, no formal semantics based on temporal logics is given to timing diagrams that is suitable for model checking.

This paper is structured as follows: Section 2 introduces a running example used throughout the rest of the paper. Section 3 introduces the language RTSTD formally. Before we give a formal semantics to RTSTD in Sect. 5 we introduce the logic TPTL in Sect. 4.

## 2 An Example Real Time STD

Throughout this paper we will use a single real time symbolic timing diagram (RTSTD) shown in Fig. 1 as a running example. It is a simplified handshaking protocol between a sender and a receiver (cf. Fig. 2). In this timing diagram part of the requirements concerning the sender are given, i.e. we are considering open systems and a modular specification style.

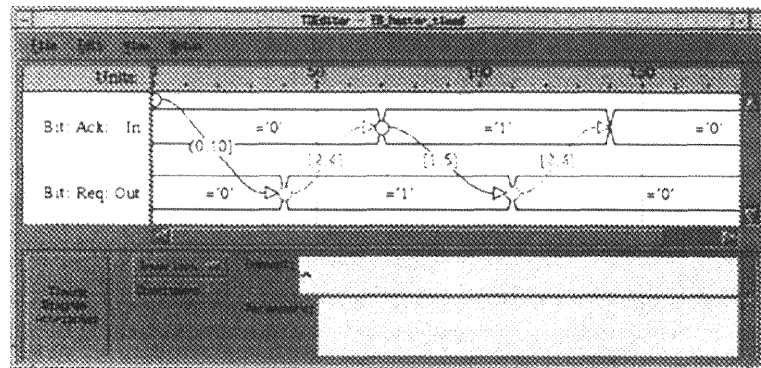


**Fig. 1.** Sender specification with real time constraints

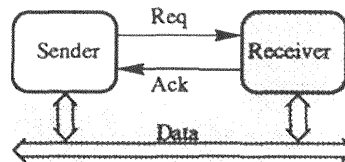


**Fig. 2.** The sender/receiver protocol. The shaded parts are not considered in the example

A RTSTD consists of four major parts:

- An *entity declaration* *ed* defining the interface of the component. It specifies the ports (or signals) (Req, Ack) of the system as well as their data type (Bit) and access mode (In, Out).
- A set of *waveforms*. Each waveform defines a sequence of values (or, more generally, a sequence of expressions) associated to the signals. The rising and falling edges in a waveform are called (*symbolic*) *events* *e*. A special *activation event* *e<sub>act</sub>* is located to the left over the waveforms.
- A set of constraints between events of (different) waveforms. We distinguish *strong constraints*, given by black solid arcs which must be satisfied by the component under consideration, and *weak constraints*, given by shaded arcs, which denote assumptions on the behaviour of the environment.
- An *activation mode* (“Invariant” or “Initial”). An *initial* diagram describes requirements on the initial system state whereas an *invariant* diagram should be satisfied again and again (see below).

The example diagram describes the following requirements specified by strong constraints (solid arcs):

- Whenever both Req and Ack are not set, then Req must be set within 10 time units (indicated by the first solid arc). Hence, at no time both Req and Ack may be continuously unset for longer than 10 time units.

- If the request is acknowledged (both Req and Ack are set) the sender must guarantee that Req is reset within 1 to 5 time units.

On the other hand, this protocol will only work correctly if the receiver that controls Ack meets the protocol definition, specified by weak constraints:

- We assume that Ack is not set before Req. Furthermore it is assumed that a request will be acknowledged within 2 to 4 time units. If Ack does not fulfill these requirements (i.e. Ack is set before Req or before two time units have elapsed, or it is not set within 4 time units after Req), we cannot continue with the normal protocol behaviour and the timing diagram is disabled, i.e. all the following requirements need not to be fulfilled.
- If both Req and Ack are set, Req is assumed to be reset first, and we assume that Ack is reset within 2 to 4 time units after request.

The activation mode of the example is “Invariant” indicating that the requirements imposed by the RTSTD must be met (the diagram is *activated*) whenever the leftmost conditions of all waveforms is matched by the actual values of the ports. In our example this happens if Req=0 and Ack=0.

### 3 Syntax of RTSTD

In this section we give a formal definition of RTSTD, guided by the graphical notation wherever possible. Furthermore we introduce the notion of *phases* and *unwinding* that will be used to define the semantics of the language.

As introduced above a RTSTD over an entity declaration *ed* consists of an activation mode  $A \in \{Initial, Invariant\}$ , an activation event  $e_{act}$ , a set of waveforms  $W$  and a set of timing constraints  $RT$ :

$$TD \stackrel{\text{def}}{=} (A, W, e_{act}, RT)$$

Each waveform  $\rho \in W$  consists of a set of *symbolic events*  $\rho_1, \rho_2, \dots$  which are linearly ordered from left to right. A topmost symbolic event  $\top_\rho$  that is not depicted in the graphical notation is used to identify that a waveform has reached its end. Thus, the waveform identified with the Ack signal consists of the linear ordered symbolic events

$$Ack_1 <_{Ack} Ack_2 <_{Ack} \top_{Ack}$$

(e.g.  $Ack_1$  is the rising edge from Ack=0 to Ack=1). With every symbolic event we identify two propositions: An *activation condition* *actc* given by the condition to the left of the event, e.g.

$$actc(Ack_1) \stackrel{\text{def}}{=} Ack=0 ,$$

and a *continuation condition* *contc* depicted to the right:<sup>1</sup>

$$\text{contc}(\text{Ack}_1) \stackrel{\text{def}}{=} \text{Ack}=1$$

We require that  $\text{contc}(\rho_i)$  is equal to the activation condition of the successor event  $\text{actc}(\rho_{i+1})$  if  $\rho_{i+1} \neq \top_\rho$  which explains why there is only one expression depicted between adjoining events:

$$\text{contc}(\text{Ack}_1) \stackrel{\text{def}}{=} \text{actc}(\text{Ack}_2) \stackrel{\text{def}}{=} \text{Ack}=1$$

Furthermore we define

$$\text{actc}(\top_\rho) = \text{contc}(\top_\rho) = \text{true} .$$

*Constraint Arcs.* The constraint arcs  $RT$  are used to demand an ordering on the symbolic events of different waveforms and to impose timing requirements on events. There are four basic classes of constraints (cf. Fig. 3).

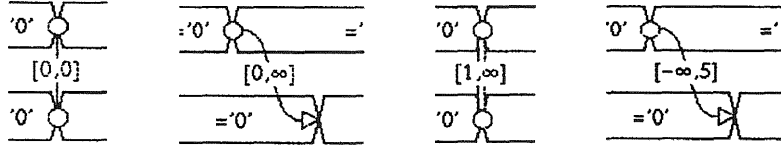


Fig. 3. Simultaneous, precedence, conflict<sub>1</sub> and leads-to<sub>5</sub> constraints

Two of them impose an ordering on the events: *Simultaneous*,  $RT^=$ , requiring that two symbolic events must be taken in the same unwinding step, and *Precedence*,  $RT^{\leq}$ , stating that the target event must not happen before the source event. The other two carry timing annotations and impose timing requirements: *Conflict*,  $RT_t^{\neq}$ , requires that the two events connected may not happen within the same step and that furthermore their occurrence times must be at least  $t$  time units apart, and *Leads-to*,  $RT_t^{\rightsquigarrow}$ , requiring that the target event may not happen later than  $t$  time units after the source event<sup>2</sup>. Each of these constraint classes come as weak or strong constraints indicating an assumed resp. required behaviour. Arcs in the graphical notation may be combined constraints. For example, the second solid arc in Fig. 1 is formed by conjunction of strong precedence, conflict<sub>1</sub> and leads-to<sub>5</sub> constraints, i.e.  $(\text{Ack}_1, \text{Req}_2) \in RT^{\leq}, RT_1^{\neq}, RT_5^{\rightsquigarrow}$ .

<sup>1</sup> In addition, the full language allows additional *exit conditions* that may be used to specify conditions that allow premature deactivation of the diagram. They are not used in the example and not considered in the rest of this paper. Refer to [9] for details.

<sup>2</sup> Note that that target event of a leads-to may well happen before the source event. No ordering is induced by a leads-to constraint.

**Phases and Unwinding.** When activated, a timing diagram restricts the behaviour of the specified system. These restrictions can be defined by sequences of *phases* derived by *unwinding* the timing diagram. Each phase  $\zeta$  is characterized by the set of symbolic events that may occur next. I.e. a phase  $\zeta$  contains exactly one symbolic event  $\rho_i$  of every waveform  $\rho \in W$ .  $\zeta$  describes the phase before one (or some) of the events  $E \subseteq \zeta$  occurs, hence the activation conditions  $actc(e), e \in \zeta$  hold throughout the phase. For example, if the RTSTD of Fig. 1 is activated the first phase is denoted by  $\{Req_1, Act_1\}$  and  $Req=0 \wedge Ack=0$  holds during this phase. If the continuation conditions  $contc$  of some events  $E \subseteq \zeta_1$  become valid we may change to a successor phase by replacing the events  $E$  that took place by their waveform successor events  $E'$ . We call this an *unwinding step* from  $\zeta_1$  to  $\zeta_2 = (\zeta_1 \setminus E) \cup E'$  of the set of waveforms  $W$  and say that  $\zeta_2$  is an  $E$ -successor of  $\zeta_1$ . If we forget about the constraint arcs for a moment (that restrict the allowed unwinding steps) the phase  $\{Ack_1, Req_1\}$  has three successor phases: We may unwind to  $\{Ack_2, Req_1\}$  ( $\{Ack_1, Req_2\}$ ,  $\{Ack_2, Req_2\}$ ) if Ack (resp. Req, or both) changes from 0 to 1. Note that this successor relation on phases induces a partial order on the phases  $\zeta_i$ . The waveform delimiters  $\top_\rho$  must never be taken since they have no successor. Thus the very last phase of  $W$  is reached if  $\zeta$  consists only of top elements  $\zeta = \{\top_{\rho_i} \mid \rho_i \in W\}$ .

## 4 Real Time Model

The formal semantics of RTSTD will be given by a translation to a timed propositional logic (TPTL, [2]) dialect that uses clocks and *reset quantification* as used e.g. in [11]. Timed automata [1] serve as an intermediate representation. We characterize a system by the sequence of observable signal changes over the time at its interface called *timed state sequence*  $tss = (\bar{\sigma}, \bar{\tau})$  [2]. A timed state sequence consists of an infinite sequence of states  $\bar{\sigma} = \sigma_0 \sigma_1 \dots$  where each  $\sigma_i$  maps each port of the component to a type correct value, and an infinite time sequence  $\bar{\tau} = \tau_0 \tau_1 \dots$  with  $\tau_i \in \mathbb{N}$  that maps each state in the sequence to a discrete time stamp. The time sequence  $\bar{\tau}$  must be monotonous ( $\forall i : \tau_i \leq \tau_{i+1}$ ) and unbounded ( $\forall n \in \mathbb{N} : \exists \tau_i : \tau_i > n$ ).

**TPTL.** Timed PTL uses the operators of the untimed logic PTL such as  $\Box\phi$  (always)  $\phi_1$  **until**  $\phi_2$  ( $\phi_2$  will hold somewhere in the future and  $\phi_1$  holds before) and  $\phi_1$  **unless**  $\phi_2$  ( $\phi_1$  holds as long as  $\phi_2$  does not hold). Arbitrary boolean expressions  $Expr_{ed}$  over the ports of the entity declaration may be used as basic propositions of the logic. On top of PTL our logic uses clocks  $z_i \in C$  that may be constrained within the formulae by clock constraints  $\Phi(C) \stackrel{\text{def}}{=} z_1 \leq t_1 \mid z_1 + t_1 \leq z_2 + t_2$  ( $t_i \in \mathbb{N}$  time constants) and reset to 0 by the *reset quantifier*  $z.\phi$ . (For the full definition and formal semantics see [2] for TPTL with freeze quantification and [9] for TPTL as used here.) For example, in

$$z.(Req=0 \wedge Ack=0 \text{ until } Req=1 \wedge Ack=0 \wedge z \leq 10)$$

the clock variable  $z$  is reset to 0 at the beginning of the formula. While we wait for request to change its value the clock increases and we require that  $z \leq 10$  when the change on request takes place (cf. Fig. 4).

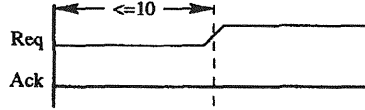


Fig. 4. Request must be set within 10 time units

## 5 Semantics of RTSTD

We define the formal semantics of RTSTD in three steps. First we translate the set of waveforms with the constraints imposed to an acyclic fragment of a timed Büchi automaton called *unwinding structure*. This unwinding structure is then translated to TPTL yielding the raw formula of the RTSTD body. In the last step we add the translation of the activation mode. The formula resulting is the characterization of the diagram in TPTL and may be used in formal proofs.

### 5.1 Unwinding Structure

The unwinding structure of a RTSTD  $td$  is a fragment of a timed Büchi automaton  $(S, s_0, C, \mathcal{T}, F)$ .

- The states  $S$  of the automaton correspond to the phases  $\zeta_i$  of  $td$  plus a unique exit state  $\zeta_{exit}$ .
- The start state  $s_0$  is the first phase of the diagram, e.g.  $\{Ack_1, Req_1\}$  in our example.
- The set of clocks  $C$  comprises one clock  $z_e$  for each symbolic event  $e$  in the waveforms of  $td$  as well as one clock  $z_{act}$  for the activation event  $e_{act}$ . If we do an unwinding step in the diagram all the clocks corresponding to the events taken in this step are reset.
- The set of accepting states  $F \subseteq S$  is derived from the set of strong leads-to constraints. If there is some event  $e_1$  that already took place connected by a strong leads-to constraint with an event  $e_2$  that has not took place yet this indicates that  $e_2$  must eventually happen. Thus we need to leave the actual phase — or in the translation the actual state — and thus need to exclude this state from  $F$ . Furthermore, the exit state  $\zeta_{exit}$  that indicates that the diagram was deactivated is in  $F$ .
- Transitions between phases correspond to unwinding steps.  $(\zeta_1, \zeta_2, cond, reset, timing)$  is in the transition relation if  $\zeta_2$  is a  $E$ -successor phase of  $\zeta_1$ , and all ordering constraint imposed by the simultaneous, precedence and conflict constraints are met.

*cond* states that for all symbolic events  $e \in E$  that happened in this unwinding *contc*( $e$ ) is met, and that the remaining events  $e' \in \zeta_1 \setminus E$  are still activated (*actc*( $e'$ )). *reset* contains all the clocks  $z_e$  associated with the events  $e \in E$  taken in this unwinding. These clocks are reset by the transitions, hence all constraints whose source is an event  $e \in E$  can be checked in the future. Finally, the timing condition *timing* must ensure that all real time constraints imposed by conflict and leads-to constraints are met.

The left half of the automaton in Fig. 5 corresponds to the unwinding steps of the diagram. Since the precedence constraint arcs impose a linear order on the symbolic events

$$Req_1 \rightarrow Ack_1 \rightarrow Req_2 \rightarrow Ack_2$$

each phase has exactly one valid successor phase.

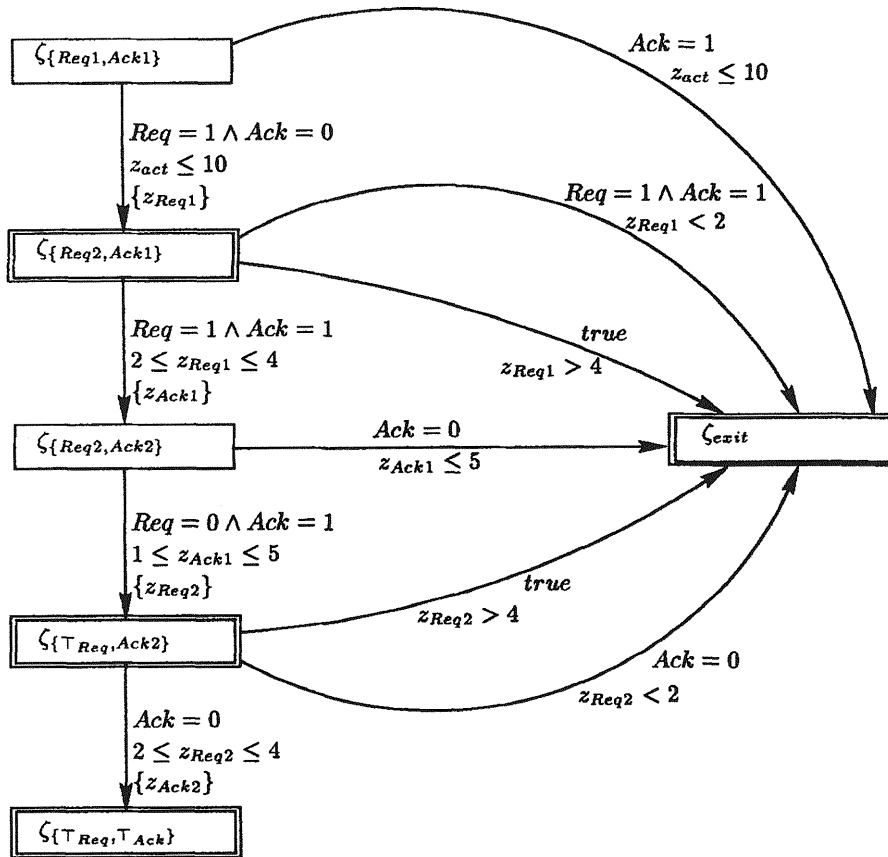


Fig. 5. The unwinding structure of the RTSTD in Fig. 1

The transition steps to  $\zeta_{exit}$  in the automaton correspond to premature deactivation of the diagram due to the violation of some weak constraint. Recall that weak constraints are used to model assumption on how the environment will behave, and strong constraints to define how the component specified must behave. Thus, if both a strong and a weak constraint are violated at the same instance there is a malfunction in both the component and the environment. The faulty behaviour of the component must be detected, i.e. we disallow an exit step in this case. This policy is achieved by the *timing* and *cond* predicates associated with the transitions:

- The first weak constraint in Fig. 1 is violated if *Ack* changes to 1 before or simultaneous with *Req*. Thus a transition from  $\zeta_{\{Req1, Ack1\}}$  to  $\zeta_{exit}$  is added in the unwinding structure. But due to the first strong constraint a failure occurs if the diagram has been activated longer than 10 time units. Thus we have to ensure  $z_{act} \leq 10$ .
- If we change to phase  $\{Req_2, Ack_1\}$  and *Ack* changes to 1 before two time units have elapsed, we may exit the diagram — but only if *Req* does not change to 0 at the same moment which causes a violation of a strong constraint.
- Timing violation of weak leads-to constraints happens if we stay in the phase longer than four time units.

This general rule that a violation of a strong constraint causes a failure despite the violation of a weak constraint has one exception. If both assumed and required upper timing bounds were not met (that is, both a strong and a weak leads-to constraint are violated) we have to take a closer look at the timing before deciding on exit or failure. Consider the example diagrams of Fig. 6 and the system behaviour of Fig. 7 which will result in a timed state sequence *tss*:

$$((x = 0, y = 0), \tau = 0), ((x = 1, y = 1), \tau = 1), ((x = 0, y = 0), \tau = 11) \dots$$

At global time 11 where we observe the unsetting of the signals both the weak and the strong constraint are violated. But in the upper diagram the weak constraint was violated earlier in time, in the lower diagram the strong constraint. The translation procedure ensures that the corresponding transitions in the unwinding structure allow a transition to  $\zeta_{exit}$  for the upper diagram but disallows it for the lower one (cf. Fig. 8) in the given situation. This is done by comparing the time elapsed since the timers  $z_{x1}$  and  $z_{y1}$  have reached their allowed upper bound.

## 5.2 Translation to TPTL

In a second translation step the unwinding structure described above is translated to PTL. Finally, the activation mode of the timing diagram is added. Translation of the unwinding structure is accomplished by the translation function *TL* on the states of the unwinding structure, starting in the first phase. We may stay in a phase as long as its activation condition *actcond* is valid, and leave

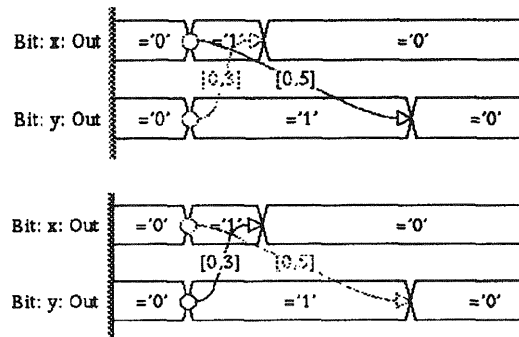


Fig. 6. Two RTSTD: The upper one assumes that  $x$  is reset within 3 time units after the setting of  $y$  and requires that  $y$  is reset within 5 time units after  $x$  is set. In the lower diagram the assumptions and requirements are exchanged.

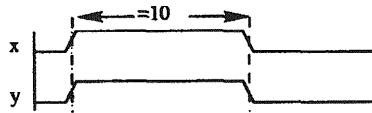


Fig. 7. Both  $x$  and  $y$  change their values from 1 to 0 after 10 time units

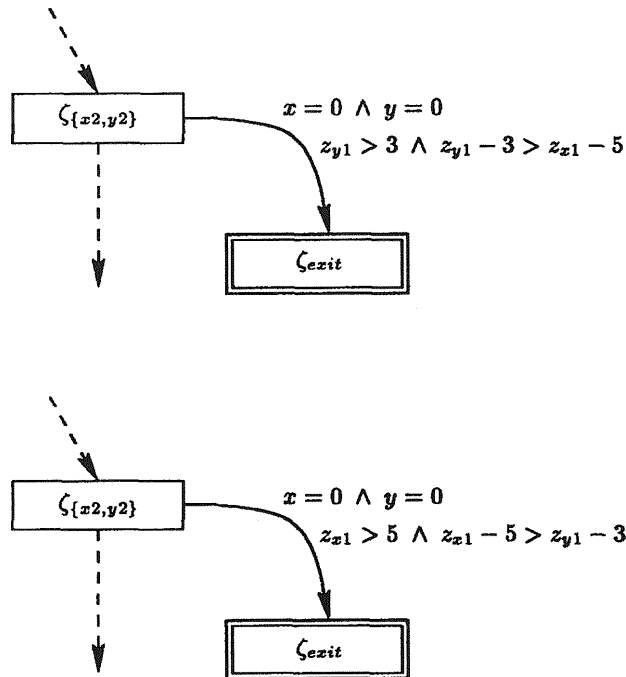


Fig. 8. Extract from the unwinding structures of the RTSTD in Fig. 6

the phase by any outgoing transition if the appropriate valuation of the ports and the imposed timing conditions hold. Hence, translation of the first state is straightforward. We may stay in phase  $\zeta_{\{Req1, Ack1\}}$  as long as *Req* and *Ack* are 0, but must finally leave the phase (since  $\zeta_{\{Req1, Ack1\}}$  is not an accepting state) by either taking the transition to  $\zeta_{exit}$  (where the STD no longer constraints the ports,  $TL(\zeta_{exit}) = true$ ) or by an unwinding step to the successor phase  $\zeta_{\{Req2, Ack1\}}$ .

$$\begin{aligned}
 TL(\zeta_{\{Req1, Ack1\}}) = & \\
 & (Req=0 \wedge Ack=0 \text{ until } (Ack=1 \wedge z_{act} \leq 10) \\
 & \quad \vee Req=1 \wedge Ack=0 \\
 & \quad \wedge z_{act} \leq 10 \wedge z_{Req1}.TL(\zeta_{\{Req2, Ack1\}}))
 \end{aligned}$$

If a state is accepting we use the **unless** operator instead of **until** since we may stay in the corresponding phase forever. Thus the translation of the remaining phases is as follows:

$$\begin{aligned}
 TL(\zeta_{\{Req2, Ack1\}}) = & \\
 & (Req=1 \wedge Ack=0 \text{ unless } (Req=1 \wedge Ack=1 \wedge z_{Req1} < 2) \\
 & \quad \vee (z_{Req1} > 4) \\
 & \quad \vee (Req=1 \wedge Ack=1 \\
 & \quad \wedge 2 \leq z_{Req1} \leq 4 \wedge z_{Ack1}.TL(\zeta_{\{Req2, Ack2\}})) \\
 TL(\zeta_{\{Req2, Ack2\}}) = & \\
 & (Req=1 \wedge Ack=1 \text{ until } (Ack=0 \wedge z_{Ack1} \leq 5) \\
 & \quad \vee (Req=0 \wedge Ack=1 \wedge \\
 & \quad \wedge 1 \leq z_{Ack1} \leq 5 \wedge z_{Req2}.TL(\zeta_{\{\neg Req, Ack2\}})) \\
 TL(\zeta_{\{\neg Req, Ack2\}}) = & \\
 & (Ack=1 \text{ unless } (Ack=0 \wedge z_{Req2} < 2) \\
 & \quad \vee (z_{Req2} > 4) \\
 & \quad \vee (Ack = 0 \wedge 2 \leq z_{Req2} \leq 4))
 \end{aligned}$$

Finally, we must add the activation mode to the formula. It resets the activation clock  $z_{act}$  and forces activation of the diagram

- either — if the activation mode is “Invariant” — whenever the leftmost conditions of the waveforms are matched by the actual values at the ports:

$$\Box(Req=0 \wedge Ack=0 \Rightarrow z_{act}.TL(\zeta_{\{Req1, Ack1\}}))$$

Observe that there is no straightforward way to express this invariance closure ( $\Box$ ) on the level of automata as there may be several invocations of the same timing diagrams activated if during unwinding the activation condition is met again.

- or, if the activation mode is “Initial” — at system startup. In this case the activation of our example would have been

$$\text{Req}=0 \wedge \text{Ack}=0 \wedge z_{act}.TL(\zeta_{\{\text{Req}1, \text{Ack}1\}}) \cdot$$

## 6 Discussion

In this paper we have extended the graphical specification language STD to allow specification of real time properties and given an introduction to the formal semantics of the extended language. Since the semantics of “pure” STD is not altered the constraints used in STD can be treated as special cases of RTSTD constraints which can be handled more efficient. Thus we only need to use real time requirements where really necessary and stick to qualitative timing wherever possible which helps decreasing complexity of the proofs.

A major obstacle of the semantics presented in this paper is the complexity of TPTL model checking. There are two answers to tackle this problem: on the one hand many subformulas produced during translation can be shared between different paths through the timing diagram which reduces the proof amount drastically. Furthermore, the number of clocks needed in the formula may be limited by static analysis of the diagram. For example, in the diagram presented a single clock would suffice in the translation.

On the other hand, when model checking synchronous (clocked) systems we will use PTL without clocks to give semantics to RTSTD. In this case, we identify each time unit in the specification with a step of the system and use PTL’s nexttime operator  $\bigcirc$  to specify time intervals.

Thus, future research will focus on extending model checking techniques based on RTSTD and investigating subclasses of RTSTD with respect to efficient proof methods.

## References

1. R. Alur and D. Dill. The theory of timed automata. In J.W. de Bakker, T. Henzinger, and W.P. de Roever, editors, *Real Time: Theorie in Practice, REX Workshop 1991*, LNCS 600, pages 45–73. Springer-Verlag, 1992.
2. R. Alur and T. Henzinger. A really temporal logic. *Journal of the ACM*, 41:181–204, 1994.
3. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL – a tool suite for automatic verification of real-time systems. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, 1995.
4. G. Boriello. Formalized timing diagrams. In *Proceedings, The European Conference on Design Automation*, pages 372–377, Brussels, Belgium, 1992.
5. J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Proceedings, 27th ACM/IEEE Design Automation Conference*, pages 46–51, June 1990.
6. W. Damm, B. Josko, and R. Schlör. Specification and verification of VHDL-based system-level hardware designs. In E. Börger, editor, *Specification and Validation Methods*, pages 331–410. Oxford University Press, 1995.

7. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III, Verification and Control*, LNCS 1066. Springer-Verlag, 1996.
8. C. Dietz. Graphical formalization of real-time requirements. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'96)*, LNCS 1135, pages 366–384. Springer-Verlag, 1996.
9. K. Feyerabend. Real time symbolic timing diagrams. Technical report, Department of Computer Science, Oldenburg University, September 1996. <http://ca.informatik.uni-oldenburg.de>.
10. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16:403–414, 1990.
11. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
12. IEEE Standard VHDL Reference Manual IEEE Std 1076-1987, 1987.
13. P.K. Khordoc, M. Dufresne, and E. Czerny. A stimulus/response system based on hierarchical timing diagrams. Technical Report 770, Universite de Montreal, 1991.
14. C. Delgado Kloos and W. Damm, editors. *Formal Methods for Hardware Verification*. Springer-Verlag, 1996.
15. L. E. Moser, P. M. Melliar-Smith, Y.S. Ramakrishna, G. Kutty, and L. K. Dillon. The real-time graphical interval logic toolset. In *Proceedings of the 8th Int. Conference on Computer Aided Verification CAV'96*, LNCS 1102. Springer-Verlag, 1996.
16. Y.S. Ramakrishna, P.M. Melliar-Smith, L.E. Moser, L.K. Dillon, and G. Kutty. Really visual temporal reasoning. In *Proceedings of 14th Real-Time Systems Symposium*, pages 262–273, December 1993.

# **EXHIBIT C**

```

<collection>
  <record xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.loc.gov/MARC21/slim">
    <leader>00000nam a22000005i 4500</leader>
    <controlfield tag="001">978-3-540-45047-4</controlfield>
    <controlfield tag="003">DE-He213</controlfield>
    <controlfield tag="005">20220115073518.0</controlfield>
    <controlfield tag="007">cr nn 008mamaa</controlfield>
    <controlfield tag="008">121227s2000 gw | s |||| 0|eng d</controlfield>
    <datafield tag="020" ind2=" " ind1=" ">
      <subfield code="a">9783540450474</subfield>
      <subfield code="9">978-3-540-45047-4</subfield>
    </datafield>
    <datafield tag="024" ind2=" " ind1="7">
      <subfield code="a">10.1007/10722167</subfield>
      <subfield code="2">doi</subfield>
    </datafield>
    <datafield tag="050" ind2="4" ind1=" ">
      <subfield code="a">QA76.758</subfield>
    </datafield>
    <datafield tag="072" ind2="7" ind1=" ">
      <subfield code="a">UMZ</subfield>
      <subfield code="2">bicssc</subfield>
    </datafield>
    <datafield tag="072" ind2="7" ind1=" ">
      <subfield code="a">COM051230</subfield>
      <subfield code="2">bisacsh</subfield>
    </datafield>
    <datafield tag="072" ind2="7" ind1=" ">
      <subfield code="a">UMZ</subfield>
      <subfield code="2">thema</subfield>
    </datafield>
    <datafield tag="082" ind2="4" ind1="0">
      <subfield code="a">005.1</subfield>
      <subfield code="2">23</subfield>
    </datafield>
    <datafield tag="245" ind2="0" ind1="1">
      <subfield code="a">Computer Aided Verification</subfield>
      <subfield code="h">[electronic resource] :</subfield>
      <subfield code="b">12th International Conference, CAV 2000 Chicago, IL,
        USA, July 15-19, 2000 Proceedings /</subfield>
      <subfield code="c">edited by E. Allen Emerson, A. Prasad
        Sistla.</subfield>
    </datafield>
    <datafield tag="250" ind2=" " ind1=" ">
      <subfield code="a">1st ed. 2000.</subfield>
    </datafield>
    <datafield tag="264" ind2="1" ind1=" ">
      <subfield code="a">Berlin, Heidelberg :</subfield>

```

```

    <subfield code="b">Springer Berlin Heidelberg :</subfield>
    <subfield code="b">Imprint: Springer,</subfield>
    <subfield code="c">2000.</subfield>
</datafield>
<datafield tag="300" ind2=" " ind1=" ">
    <subfield code="a">X, 590 p.</subfield>
    <subfield code="b">online resource.</subfield>
</datafield>
<datafield tag="336" ind2=" " ind1=" ">
    <subfield code="a">text</subfield>
    <subfield code="b">txt</subfield>
    <subfield code="2">rdacontent</subfield>
</datafield>
<datafield tag="337" ind2=" " ind1=" ">
    <subfield code="a">computer</subfield>
    <subfield code="b">c</subfield>
    <subfield code="2">rdamedia</subfield>
</datafield>
<datafield tag="338" ind2=" " ind1=" ">
    <subfield code="a">online resource</subfield>
    <subfield code="b">cr</subfield>
    <subfield code="2">rdacarrier</subfield>
</datafield>
<datafield tag="347" ind2=" " ind1=" ">
    <subfield code="a">text file</subfield>
    <subfield code="b">PDF</subfield>
    <subfield code="2">rda</subfield>
</datafield>
<datafield tag="490" ind2=" " ind1="1">
    <subfield code="a">Lecture Notes in Computer Science,</subfield>
    <subfield code="x">1611-3349 ;</subfield>
    <subfield code="v">1855</subfield>
</datafield>
<datafield tag="505" ind2=" " ind1="0">
    <subfield code="a">Invited Talks and Tutorials -- Keynote Address:
    Abstraction, Composition, Symmetry, and a Little Deduction: The Remedies
    to State Explosion -- Invited Address: Applying Formal Methods to
    Cryptographic Protocol Analysis -- Invited Tutorial: Boolean
    Satisfiability Algorithms and Applications in Electronic Design Automation
    -- Invited Tutorial: Verification of Infinite-state and Parameterized
    Systems -- Regular Papers -- An Abstraction Algorithm for the Verification
    of Generalized C-Slow Designs -- Achieving Scalability in Parallel
    Reachability Analysis of Very Large Circuits -- An Automata-Theoretic
    Approach to Reasoning about Infinite-State Systems -- Automatic
    Verification of Parameterized Cache Coherence Protocols -- Binary
    Reachability Analysis of Discrete Pushdown Timed Automata -- Boolean
    Satisfiability with Transitivity Constraints -- Bounded Model Construction
    for Monadic Second-Order Logics -- Building Circuits from Relations --
    Combining Decision Diagrams and SAT Procedures for Efficient Symbolic

```

Model Checking -- On the Completeness of Compositional Reasoning -- Counterexample-Guided Abstraction Refinement -- Decision Procedures for Inductive Boolean Functions Based on Alternating Automata -- Detecting Errors Before Reaching Them -- A Discrete Strategy Improvement Algorithm for Solving Parity Games -- Distributing Timed Model Checking -- How the Search Order Matters -- Efficient Algorithms for Model Checking Pushdown Systems -- Efficient Büchi Automata from LTL Formulae -- Efficient Detection of Global Properties in Distributed Systems Using Partial-Order Methods -- Efficient Reachability Analysis of Hierarchical Reactive Machines -- Formal Verification of VLIW Microprocessors with Speculative Execution -- Induction in Compositional Model Checking -- Liveness and Acceleration in Parameterized Verification -- Mechanical Verification of an Ideal Incremental ABR Conformance Algorithm -- Model Checking Continuous-Time Markov Chains by Transient Analysis -- Model-Checking for Hybrid Systems by Quotienting and Constraints Solving -- Prioritized Traversal: Efficient Reachability Analysis for Verification and Falsification -- Regular Model Checking -- Symbolic Techniques for Parametric Reasoning about Counter and Clock Systems -- Syntactic Program Transformations for Automatic Abstraction -- Temporal-logic Queries -- Are Timed Automata Updatable? -- Tuning SAT Checkers for Bounded Model Checking -- Unfoldings of Unbounded Petri Nets -- Verification Diagrams Revisited: Disjunctive Invariants for Easy Verification -- Verifying Advanced Microarchitectures that Support Speculation and Exceptions -- Tool Papers -- FoCs - Automatic Generation of Simulation Checkers from Formal Specifications -- IF: A Validation Environment for Timed Asynchronous Systems -- Integrating WS1S with PVS -- PET: An Interactive Software Testing Tool -- A Proof-Carrying Code Architecture for Java -- The StateMate Verification Environment -- TAPS: A First-Order Verifier for Cryptographic Protocols -- VINAS-P: A Tool for Trace Theoretic Verification of Timed Asynchronous Circuits -- XMC: A Logic-Programming-Based Verification Toolset.

</datafield>

<datafield tag="520" ind2=" " ind1=" ">

<subfield code="a">This volume contains the proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000) held in Chicago, Illinois, USA during 15-19 July 2000. The CAV conferences are devoted to the advancement of the theory and practice of formal methods for hardware and software verification. The conference covers the spectrum from theoretical foundations to concrete applications, with an emphasis on verification algorithms, methods, and tools together with techniques for their implementation. The conference has traditionally drawn contributions from both researchers and practitioners in academia and industry. This year 91 regular research papers were submitted out of which 35 were accepted, while 14 brief tool papers were submitted, out of which 9 were accepted for presentation. CAV included two invited talks and a panel discussion. CAV also included a tutorial day with two invited tutorials. Many industrial companies have shown a serious interest in CAV, ranging from using the presented technologies in their business to developing and marketing their own formal verification tools. We are very proud of the

support we receive from industry. CAV 2000 was sponsored by a number of generous and forward-looking companies and organizations including: Cadence Design Systems, IBM Research, Intel, Lucent Technologies, Mentor Graphics, the Minerva Center for Verification of Reactive Systems, Siemens, and Synopsys. The CAV conference was founded by its Steering Committee: Edmund Clarke (CMU), Bob Kurshan (Bell Labs), Amir Pnueli (Weizmann), and Joseph Sifakis (Verimag).

Software engineering.

Computer science.

Machine theory.

Computers, Special purpose.

Artificial intelligence.

Software Engineering.

Computer Science Logic and Foundations of Programming.

Formal Languages and Automata Theory.

Special Purpose and Application-Based Systems.

Artificial Intelligence.

Emerson, E. Allen.

editor.

edt

<http://id.loc.gov/vocabulary/relators/edt>

Sistla, A. Prasad.

```

    <subfield code="e">editor.</subfield>
    <subfield code="4">edt</subfield>
    <subfield code="4">http://id.loc.gov/vocabulary/relators/edt</subfield>
</datafield>
<datafield tag="710" ind2=" " ind1="2">
  <subfield code="a">SpringerLink (Online service)</subfield>
</datafield>
<datafield tag="773" ind2=" " ind1="0">
  <subfield code="t">Springer Nature eBook</subfield>
</datafield>
<datafield tag="776" ind2="8" ind1="0">
  <subfield code="i">Printed edition:</subfield>
  <subfield code="z">9783662186701</subfield>
</datafield>
<datafield tag="776" ind2="8" ind1="0">
  <subfield code="i">Printed edition:</subfield>
  <subfield code="z">9783540677703</subfield>
</datafield>
<datafield tag="830" ind2="0" ind1=" ">
  <subfield code="a">Lecture Notes in Computer Science,</subfield>
  <subfield code="x">1611-3349 ;</subfield>
  <subfield code="v">1855</subfield>
</datafield>
<datafield tag="856" ind2="0" ind1="4">
  <subfield code="u">https://doi.org/10.1007/10722167</subfield>
</datafield>
<datafield tag="912" ind2=" " ind1=" ">
  <subfield code="a">ZDB-2-SCS</subfield>
</datafield>
<datafield tag="912" ind2=" " ind1=" ">
  <subfield code="a">ZDB-2-SXCS</subfield>
</datafield>
<datafield tag="912" ind2=" " ind1=" ">
  <subfield code="a">ZDB-2-LNC</subfield>
</datafield>
<datafield tag="912" ind2=" " ind1=" ">
  <subfield code="a">ZDB-2-BAE</subfield>
</datafield>
<datafield tag="950" ind2=" " ind1=" ">
  <subfield code="a">Computer Science (SpringerNature-11645)</subfield>
</datafield>
<datafield tag="950" ind2=" " ind1=" ">
  <subfield code="a">Computer Science (R0) (SpringerNature-43710)</subfield>
</datafield>
</record>
</collection>

```

# **EXHIBIT D**

```

<collection>
  <record xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.loc.gov/MARC21/slim">
    <leader>00000nam a22000005i 4500</leader>
    <controlfield tag="001">978-3-540-69058-0</controlfield>
    <controlfield tag="003">DE-He213</controlfield>
    <controlfield tag="005">20220115085803.0</controlfield>
    <controlfield tag="007">cr nn 008mamaa</controlfield>
    <controlfield tag="008">121227s1997 gw | s |||| 0|eng d</controlfield>
    <datafield tag="020" ind2=" " ind1=" ">
      <subfield code="a">9783540690580</subfield>
      <subfield code="9">978-3-540-69058-0</subfield>
    </datafield>
    <datafield tag="024" ind2=" " ind1="7">
      <subfield code="a">10.1007/3-540-63010-4</subfield>
      <subfield code="2">doi</subfield>
    </datafield>
    <datafield tag="050" ind2="4" ind1=" ">
      <subfield code="a">QA76.758</subfield>
    </datafield>
    <datafield tag="072" ind2="7" ind1=" ">
      <subfield code="a">UMZ</subfield>
      <subfield code="2">bicssc</subfield>
    </datafield>
    <datafield tag="072" ind2="7" ind1=" ">
      <subfield code="a">COM051230</subfield>
      <subfield code="2">bisacsh</subfield>
    </datafield>
    <datafield tag="072" ind2="7" ind1=" ">
      <subfield code="a">UMZ</subfield>
      <subfield code="2">thema</subfield>
    </datafield>
    <datafield tag="082" ind2="4" ind1="0">
      <subfield code="a">005.1</subfield>
      <subfield code="2">23</subfield>
    </datafield>
    <datafield tag="245" ind2="0" ind1="1">
      <subfield code="a">Transformation-Based Reactive Systems
        Development</subfield>
      <subfield code="h">[electronic resource] :</subfield>
      <subfield code="b">4th International AMAST Workshop on Real-Time Systems
        and Concurrent and Distributed Software, ARTS'97, Palma, Mallorca, Spain,
        May 21 - 23, 1997, Proceedings /</subfield>
      <subfield code="c">edited by Miquel Bertran, Teodor Rus.</subfield>
    </datafield>
    <datafield tag="250" ind2=" " ind1=" ">
      <subfield code="a">1st ed. 1997.</subfield>
    </datafield>
    <datafield tag="264" ind2="1" ind1=" ">

```

```

    <subfield code="a">Berlin, Heidelberg :</subfield>
    <subfield code="b">Springer Berlin Heidelberg :</subfield>
    <subfield code="b">Imprint: Springer,</subfield>
    <subfield code="c">1997.</subfield>
  </datafield>
  <datafield tag="300" ind2=" " ind1=" ">
    <subfield code="a">XI, 436 p.</subfield>
    <subfield code="b">online resource.</subfield>
  </datafield>
  <datafield tag="336" ind2=" " ind1=" ">
    <subfield code="a">text</subfield>
    <subfield code="b">txt</subfield>
    <subfield code="2">rdacontent</subfield>
  </datafield>
  <datafield tag="337" ind2=" " ind1=" ">
    <subfield code="a">computer</subfield>
    <subfield code="b">c</subfield>
    <subfield code="2">rdamedia</subfield>
  </datafield>
  <datafield tag="338" ind2=" " ind1=" ">
    <subfield code="a">online resource</subfield>
    <subfield code="b">cr</subfield>
    <subfield code="2">rdacarrier</subfield>
  </datafield>
  <datafield tag="347" ind2=" " ind1=" ">
    <subfield code="a">text file</subfield>
    <subfield code="b">PDF</subfield>
    <subfield code="2">rda</subfield>
  </datafield>
  <datafield tag="490" ind2=" " ind1="1">
    <subfield code="a">Lecture Notes in Computer Science,</subfield>
    <subfield code="x">1611-3349 ;</subfield>
    <subfield code="v">1231</subfield>
  </datafield>
  <datafield tag="505" ind2=" " ind1="0">
    <subfield code="a">What was Llull up to? -- Llull as computer scientist or
    why Llull was one of us -- Deductive verification of real-time systems
    using STeP -- Refinement of Time -- The verus language: Representing time
    efficiently with BDDs -- Refining interval temporal logic specifications
    -- Integrating temporal logics and model checking algorithms --
    PLC-automata: A new class of implementable real-time automata --
    Communication concepts for statecharts: A semantic foundation -- Regular
    processes and timed automata -- A visual formalism for real time
    requirement specifications -- Formal specification and verification method
    of concurrent and distributed systems by restricted timed automata --
    Transformational formal development of real-time systems -- A
    transformation of monitor into communication synchronized parallel
    processes: A systematic refinement step in design -- Contracts for ODP --
    Affine transformations in Signal and their application in the

```

specification and validation of real-time systems -- Action-based concurrency and synchronization for objects -- Communication Extended Abstract Types in the refinement of parallel communicating processes -- Verification and refinement of distributed programs in a fair framework -- Formalizing real-time scheduling as program refinement -- Specification and refinement of continuous real-time systems -- High-level execution time analysis -- A sound and complete proof system for probabilistic processes -- Testing semantics for a probabilistic-timed process algebra -- Denotational semantics for timed testing -- Extending LOTOS with time: A true concurrency perspective -- of a suspend/resume operator in ET-LOTOS -- Specification and verification of a real-time field bus with formal description languages.</subfield>

</datafield>

<datafield tag="520" ind2=" " ind1=" ">

<subfield code="a">This book constitutes the refereed proceedings of the Fourth International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software, ARTS'97, held in Palma de Mallorca, Spain, in May 1997. The volume presents 24 carefully selected revised full papers. Also included are two historical contributions honoring Ramon Llull, who was born on Mallorca, as well as two invited papers. All current issues in the field of formal methods for real-time systems and distributed and concurrent systems are addressed.</subfield>

</datafield>

<datafield tag="650" ind2="0" ind1=" ">

<subfield code="a">Software engineering.</subfield>

</datafield>

<datafield tag="650" ind2="0" ind1=" ">

<subfield code="a">Computer science.</subfield>

</datafield>

<datafield tag="650" ind2="0" ind1=" ">

<subfield code="a">Computer programming.</subfield>

</datafield>

<datafield tag="650" ind2="0" ind1=" ">

<subfield code="a">Computer networks .</subfield>

</datafield>

<datafield tag="650" ind2="0" ind1=" ">

<subfield code="a">Computers, Special purpose.</subfield>

</datafield>

<datafield tag="650" ind2="4" ind1="1">

<subfield code="a">Software Engineering.</subfield>

</datafield>

<datafield tag="650" ind2="4" ind1="2">

<subfield code="a">Theory of Computation.</subfield>

</datafield>

<datafield tag="650" ind2="4" ind1="2">

<subfield code="a">Programming Techniques.</subfield>

</datafield>

<datafield tag="650" ind2="4" ind1="2">

<subfield code="a">Computer Communication Networks.</subfield>

```

</datafield>
<datafield tag="650" ind2="4" ind1="2">
  <subfield code="a">Special Purpose and Application-Based
    Systems.</subfield>
</datafield>
<datafield tag="700" ind2=" " ind1="1">
  <subfield code="a">Bertran, Miquel.</subfield>
  <subfield code="e">editor.</subfield>
  <subfield code="4">edt</subfield>
  <subfield code="4">http://id.loc.gov/vocabulary/relators/edt</subfield>
</datafield>
<datafield tag="700" ind2=" " ind1="1">
  <subfield code="a">Rus, Teodor.</subfield>
  <subfield code="e">editor.</subfield>
  <subfield code="4">edt</subfield>
  <subfield code="4">http://id.loc.gov/vocabulary/relators/edt</subfield>
</datafield>
<datafield tag="710" ind2=" " ind1="2">
  <subfield code="a">SpringerLink (Online service)</subfield>
</datafield>
<datafield tag="773" ind2=" " ind1="0">
  <subfield code="t">Springer Nature eBook</subfield>
</datafield>
<datafield tag="776" ind2="8" ind1="0">
  <subfield code="i">Printed edition:</subfield>
  <subfield code="z">9783662196694</subfield>
</datafield>
<datafield tag="776" ind2="8" ind1="0">
  <subfield code="i">Printed edition:</subfield>
  <subfield code="z">9783540630104</subfield>
</datafield>
<datafield tag="830" ind2="0" ind1=" ">
  <subfield code="a">Lecture Notes in Computer Science,</subfield>
  <subfield code="x">1611-3349 ;</subfield>
  <subfield code="v">1231</subfield>
</datafield>
<datafield tag="856" ind2="0" ind1="4">
  <subfield code="u">https://doi.org/10.1007/3-540-63010-4</subfield>
</datafield>
<datafield tag="912" ind2=" " ind1=" ">
  <subfield code="a">ZDB-2-SCS</subfield>
</datafield>
<datafield tag="912" ind2=" " ind1=" ">
  <subfield code="a">ZDB-2-SXCS</subfield>
</datafield>
<datafield tag="912" ind2=" " ind1=" ">
  <subfield code="a">ZDB-2-LNC</subfield>
</datafield>
<datafield tag="912" ind2=" " ind1=" ">

```

```
<subfield code="a">ZDB-2-BAE</subfield>
</datafield>
<datafield tag="950" ind2=" " ind1=" ">
  <subfield code="a">Computer Science (SpringerNature-11645)</subfield>
</datafield>
<datafield tag="950" ind2=" " ind1=" ">
  <subfield code="a">Computer Science (R0) (SpringerNature-43710)</subfield>
</datafield>
</record>
</collection>
```