Scalable Compatibility for Embedded Real-Time components via Language Progressive Timed Automata

Stefan Neumann, Holger Giese

Technische Berichte Nr. 65

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 65

Stefan Neumann | Holger Giese

Scalable Compatibility for Embedded Real-Time Components via Language Progressive Timed Automata

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.de/ abrufbar.

Universitätsverlag Potsdam 2013
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2013/6385/
URN urn:nbn:de:kobv:517-opus-63853
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-63853

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-226-1

Abstract

The proper composition of independently developed components of an embedded real-time system is complicated due to the fact that besides the functional behavior also the non-functional properties and in particular the timing have to be compatible. Nowadays related compatibility problems have to be addressed in a cumbersome integration and configuration phase at the end of the development process, that in the worst case may fail. Therefore, a number of formal approaches have been developed, which try to guide the upfront decomposition of the embedded real-time system into components such that integration problems related to timing properties can be excluded and that suitable configurations can be found. However, the proposed solutions require a number of strong assumptions that can be hardly fulfilled or the required analysis does not scale well. In this paper an approach based on timed automata is represented that can provide the required guarantees for the later integration without strong assumptions, which are difficult to match in practice. The approach provides a modular reasoning scheme that permits to establish the required guarantees for the integration employing only local checks and therefore scales. It is also possible to determine potential configuration settings by means of timed game synthesis.

Contents

1 Introduction

The proper composition of independently developed components of an embedded real-time system is complicated due to the fact that besides the functional behavior also the non-functional properties and in particular the timing have to be compatible. Nowadays related compatibility problems have to be addressed in a cumbersome integration and configuration phase that in the worst case may fail. Therefore, real-time models are employed to find and check upfront a proper decomposition into components and related interfaces such that as much as possible problems during the integration can be excluded (cf. [12]).

In particular formal approaches have been developed, which try to support the upfront planning and checking of the embedded real-time systems such that integration problems at the non-functional level can be excluded. Any upfront planning and checking of the decomposition of an embedded real-time system has to be compatible with a number of requirements to be applicable in practice: (R1) As the architectures for embedded real-time systems can be rather large and include a huge number of components, the modeling and checking must be scalable. (R2) To be further appropriate for embedded real-time systems, it must be possible taking into account that the resources, which can be employed to realize the components, are limited due to economical reasons. (R3) Moreover, a network of OEMs and suppliers often develops embedded real-time systems, as in the case of automotive systems. Therefore, the modeling and checking must be possible without disclosing intellectual properties (IPs)[1] that should remain hidden, e.g., in the case they are related to implementation details. (R4) A reasoning scheme has to be expressive enough such that complex architectures containing cyclic communication dependencies and mode-dependent component interactions are supported.

1.1 State of the Art

Different approaches exist that are able to fulfill individual requirements related to (R1), (R2), (R3) or (R4). According to [2, 1] component-based approaches can be distinguish between input-universal resp. *pessimistic* approaches that allow the environment to behave arbitrarily, and input-existential interpretations resp. *optimistic* approaches that only require an environment to exist that behaves sufficiently helpful. Optimistic frameworks as described in [3] use timed automata as timed interfaces and check whether an environment exists that behaves helpful. They do only protect intellectual properties (R3) to a limited extent as the interface behavior reveals many implementation details. Furthermore, the required checks do not scale for larger architectures (R1) because all interfaces are analyzed at once or in form of combined interfaces. Any analysis needs to be applied using the combined interface and as a consequence during the analysis all relevant information of the overall architecture need to be provided, rarely allowing protecting IPs (R3). In contrast, pessimistic frameworks as in [16, 10] use conditions on the timed input/output automata (TIOA) such as progressive or input enabled behavior for ensuring that components can be safely used in an environment

[1] Properties related to details that have an economical value.

that behaves arbitrarily.[2] A progressive or input enabled timed automaton is able to receive each and every input signal at any state. The conditions are further compositional such that the composition of two progressive, deadlock and zeno free TIOA is safe by resulting in a progressive, deadlock and zeno free TIOA. Due to its compositional nature, the approaches scale well (R1) and as the conditions are rather generic the IP is also well protected (R3). However, these generic conditions result in high resource requirements. For being able to fulfill the required property of progressive or input enabled behavior, such a component must be able to receive an arbitrary number of signals (e.g., 1, 1000, 1 million or even more) in any time period greater than 0. As a consequence nearly unlimited resources need to be assumed for processing received signals. Thus, requirement (R2) is difficult or even impossible to match in practice when requiring progressive behavior.

Another related direction is the timed interface approach for data-flow based architectures as proposed in [22], which focuses only on the load that the components can handle. In its basic form, these approaches scale (R1) and can take resource restrictions into account (R2), but require acyclic architectures and do not support mode-dependent behavior (R4).[3] Behavior, where different modes can become active during runtime, realizing a different timing behavior, are not directly supported.[4] Dependencies between the data-flow as well as system characteristics as in case of the used scheduling exist, resulting in limitations for the structure of the component-based architecture. As a result, requirement (R4) is not fully supported. Existing extensions permit to cover cycles to some extend [15] as well as components described by timed automata [17], but mode-dependent component interaction at the interface is still not covered. Furthermore, the load characteristics are sufficiently abstract such that the IP remains hidden (R3).

Behavior, Interaction, Priority (BIP) [6] is a framework for the component-based construction of real-time embedded system. BIP provides mechanisms for supporting a correctness-by-construction approach and creating hierarchical architectures composed of atomic components as well as tooling for analyzing the reachable state space using explorative techniques. Concerning (R1) holds that the original BIP approach does not scale, but in [20] it is shown how to apply an analysis in a modular fashion. For being able to apply a modular analysis, deterministic behavior is required and as a result only restricted types of timed automata can be used for verification purposes. While protecting IP is not addressed (R3), BIP is expressive enough to partially cover practical cases to some extend (R4), because hierarchical architectures and cyclic ones are supported.

Another approach supporting the modular verification of timed automata that scales (R1) can be found in [11]. Like in the case of [13] (extension of BIP) only deterministic behavior is allowed. We believe that allowing only deterministic TA is much too restrictive, due to the fact that nondeterminism often is related to parameters that can be configured during and not before composition (like in the case of execution orders of constituent parts of a component that need to fit to the order of incoming signals). Further, how to support cyclic architectures (R4) using deterministic frameworks in an analysis that scales is still an open issue. How to maintain protecting IPs (R3) is not considered in [11] and [13].

[2] Only deadlocks and zeno behavior needs to be excluded for the environment.

[3] Minimal and maximal arrival times are fix.

[4] Such a mode dependent behavior can also be found in the used application example described in Sec. 3.

1.2 Contribution

In this paper, an approach is represented based on timed automata that can provide the required guarantees (R1), (R2), (R3) and (R4) for the later integration, which does not require strong assumptions difficult to match in practice as in the case of progressive, deterministic or input enabled behavior. It can be seen as a compromise between the too pessimistic approaches that avoid any reliance on the characteristics of the environment, leading to arbitrary high resource requirements (R2), and the too optimistic approaches that rely too much on helpful environment characteristics, which do not scale (R1). The introduced reasoning scheme is expressive enough such that complex architectures containing cyclic communication dependencies and mode-dependent component interactions are supported (R4).

As an example, Figure 1 shows the composition of the three components *FuelSensor*, *FuelController* and *Engine-Model* realizing the fuel rate control of a combustion engine. This architecture contains cyclic communication dependencies.[5] *FuelSensor* is responsible for the evaluation of sensor values from the engine. *FuelController* is responsible for calculating the desired fuel rate for the engine. The component *EngineModel* represents relevant behavior of the engine. In Sec. 3 this application example is discussed in more detail. For such an engine control system holds that in the domain of automotive systems only limited resource are available (R2). IPs (R3) related to implementation details of individual components often have to be protected in the case of a distributed development process. In the remainder of this work the framework of language progressive TIOA is introduced. Furthermore, it is shown how to address non-functional properties in case of the timing, for an architecture as depicted in Figure 1, such that requirements (R1), (R2), (R3) and (R4) are fulfilled.

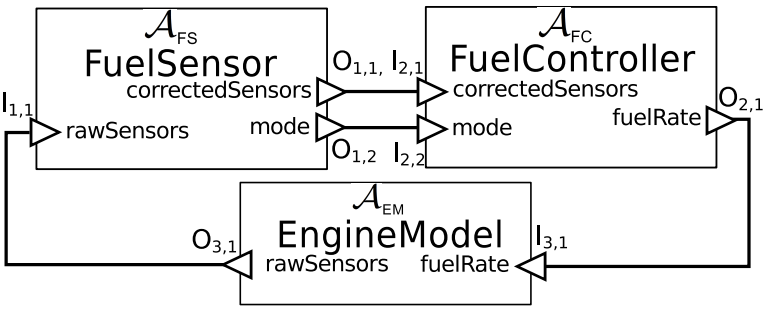


Figure 1: Fuel-Rate-Control architecture for the control of a combustion engine.

The paper is organized as follows: In Section 2, preliminary definitions like in the case of timed (IO) automata are introduced. In Section 3 the application example of the fuel rate control, like depicted in Fig. 1, is described in more detail. We show in Sec. 4 that the existing approach of progressive and receptive behavior (cf. [16]) does not fulfill requirement (R2). In Section 5 a formal definition of language progressive and language receptive TIOA is given, in contrast to the approach described in [16], fulfilling requirement (R2). Further it is discussed how to automatically check whether the property of language progressive be-

[5] The example is derived from an existing demo application shipped with the tool SystemDesk, a professional tool for designing component based architectures for real-time embedded systems (see http://www.dspace.com/systemdesk).

havior is fulfilled. Based on the defined property of language progressive timed automata, we present a reasoning scheme in Section 6. Well-formedness criteria are used building a framework for the component-based development of real-time embedded and resource restricted (R2) systems consisting of components with a single input and a single output port. How to overcome the limitation to be restricted to components with a single input/output port is shown in Sec. 7. A compositional reasoning scheme is described supporting the construction of components with multiple input/output ports and architectures containing cyclic communication dependencies (R4). In Sec. 8 the defined reasoning scheme is applied on the application example. It is shown that the created reasoning scheme scales and as a consequence fulfills requirement (R1). In Sec. 9 it is briefly discussed how existing abstraction techniques for timed automata fit into the developed approach, supporting requirement (R3). We close the paper with conclusions and an outlook on future work.

2 Prerequisites

Following the formal model of Alur-Dill style timed automata (TA, cf. [4]) is introduced. Further, the formal model of timed input/output automata (TIOA, cf. [16]) is introduced, which is the basis for the subsequently introduced class of TA.

2.1 Timed Automata

In the following, the formal model of timed automata (TA) is recalled. TA are considered that distinguish explicitly between variables in form of so-called *clocks*, respectively attributes and in form of *locations*. In the following, let $\mathbb{R}$ denote the set of reals, $\mathbb{R}^+$ denote the set of non-negative reals and $\mathbb{N}^+$ denote the set of non-negative natural numbers.

Definition 1 (Attributed Timed Automaton)
An attributed timed automaton (TA) is a tuple $A = (\Sigma, \mathcal{L}, \mathcal{L}^0, X, V, \mathcal{I}, E)$ *consisting of a finite set of signals* Σ*, which includes the empty signal* $\sigma_\epsilon \in \Sigma$*, a finite set of locations* $\mathcal{L}$*, a finite set of initial locations* $\mathcal{L}^0 \subseteq \mathcal{L}$*, a finite set of clock variables* $X = \{x_1, \ldots, x_n\}$*, a finite set of attribute variables* $V = \{v_1, \ldots, v_m\}$*, a function* $\mathcal{I} : \mathcal{L} \rightarrow 2^{\mathcal{C}(X)}$ *assigning to each location a set of clock invariants* $\mathcal{I}(l) \subseteq \mathcal{C}(X)$ *and a set of edges* $E \subseteq \mathcal{L} \times \Sigma \times 2^{\mathcal{C}(X)} \times 2^{\mathcal{C}(V)} \times 2^X \times 2^V \times \mathcal{L}$*.* $\mathcal{C}(X)$ *is the set of (clock) conditions over* X *and consists of all equations of the form* $x \sim c$ *where* $\sim \in \{\leq, \geq\}$ *and* $c \in \mathbb{N}^+$ *is a constant.* $\mathcal{C}(V)$ *is the set of (attribute variable) conditions over* V*, consisting of all equations of the form* $v \backsim r$ *where* $\backsim \in \{\leq, =, \neq, \geq\}$ *and* $r \in \mathbb{R}$*.*

An edge $e \in E$ from location l to l' is defined by a tuple $e = (l, a, \varphi, \lambda, l')$, where $a \in \Sigma$ is a signal, $\varphi \subseteq \mathcal{C}(X) \cup \mathcal{C}(V)$ is a guard that must be enabled to fire the edge, and $\lambda \subseteq 2^X \cup 2^V$. 2^X is a set of clock variables that are reset to the value 0 if the edge is taken. 2^V is a set of all assignments of the form $v \simeq n$ with $v \in V$ and $\simeq \in \{+, -, :=\}$ and n being either $n \in V$ or $n \in \mathbb{R}$.

An example of a TA is shown in Fig. 3 on page 12. *Init* is an initial location with invariant $p3 \leq 4$ over clock variable $p3$. This invariant allows the TA to stay in the initial location for at most 4 time units. The edge from location *Init* to location *CRV* contains the clock reset $d3 = 0$ of the clock $d3$. The edge from location *CRC* to location *wait* contains the signal *raw*, which need to be synchronized when taking the edge. Further, the edge contains the guard $d3 \geq 0$, allowing taking the edge only if the clock $d3$ has a value greater or equal to 0.[6] Following different types of locations are defined in case of *urgent* and *committed* locations. An urgent location of a TA is a location where time is not allowed to pass. A committed location is a location where time is not allowed to pass and, when being in a committed location, the next edge needs to be taken is an edge starting from a committed location.[7]

Definition 2 (Extended Timed Automaton)
An extended timed automaton (TA), including urgent and committed locations is a TA $A = (\Sigma, \mathcal{L}, \mathcal{UL}, \mathcal{CL}, \mathcal{L}^0, X, V, \mathcal{I}, E)$ *according to Def. 1 with* $\mathcal{L}$ *being divided into a set* $\mathcal{UL}$ *of urgent locations and a set* $\mathcal{CL}$ *of committed locations with* $\mathcal{UL} \cup \mathcal{CL} = \mathcal{L}$*. For an extended TA holds* $\mathcal{UL} \cap \mathcal{CL} = \emptyset$ *and* $\{\mathcal{UL} \cup \mathcal{CL} \cup \mathcal{L}^0\} \subseteq \mathcal{L}$*.*

Remark, an extended TA with $\mathcal{UL} = \mathcal{CL} = \emptyset$ is equivalent to an attributed TA. Following the semantics of the composition of attributed as well as extended TA is defined.

Definition 3 (Parallel composition of TA)
Let $A_1 = (\Sigma_1, \mathcal{L}_1, \mathcal{L}_1^0, X_1, V_1, \mathcal{I}_1, E_1)$ *and* $A_2 = (\Sigma_2, \mathcal{L}_2, \mathcal{L}_2^0, X_2, V_2, \mathcal{I}_2, E_2)$ *be two timed automata. Their* parallel composition $A_1 \parallel A_2$ *is defined as the timed automaton*

$$A = (\Sigma_1 \cup \Sigma_2, \mathcal{L}_1 \times \mathcal{L}_2, \mathcal{L}_1^0 \times \mathcal{L}_2^0, X_1 \cup X_2, V_1 \cup V_2, \mathcal{I}, E)$$

where $\mathcal{I}(\langle l_1, l_2 \rangle) = \mathcal{I}_1(l_1) \cup \mathcal{I}_2(l_2)$ *for all* $l_1 \in \mathcal{L}_1$*,* $l_2 \in \mathcal{L}_2$*, and* $(\langle l_1, l_2 \rangle, a, \varphi, \lambda, \langle l_1', l_2' \rangle) \in E$ *if one of the following conditions holds:*

1. $a \in \Sigma_1 \setminus \Sigma_2$*,* $l_2 = l_2'$ *and there exists* $(l_1, a, \varphi, \lambda, l_1') \in E_1$*,*
2. $a \in \Sigma_2 \setminus \Sigma_1$*,* $l_1 = l_1'$ *and there exists* $(l_2, a, \varphi, \lambda, l_2') \in E_2$*,*
3. $a \neq \sigma_\epsilon \wedge a \in \Sigma_1 \cap \Sigma_2$*, and there exist* $(l_1, a, \varphi_1, \lambda_1, l_1') \in E_1$*,* $(l_2, a, \varphi_2, \lambda_2, l_2') \in E_2$ *with* $\varphi = \varphi_1 \cup \varphi_2$*,* $\lambda = \lambda_1 \cup \lambda_2$*.*

Accordingly the parallel product of two extended timed automata is defined as follows. The only difference is, that edges leaving committed locations prevent edges leaving non-committed locations.

Definition 4 (Parallel composition of extended TA)
Let $A_1 = (\Sigma_1, \mathcal{L}_1, \mathcal{UL}_1, \mathcal{CL}_1, \mathcal{L}_1^0, X_1, V_1, \mathcal{I}_1, E_1)$ *and* $A_2 = (\Sigma_2, \mathcal{L}_2, \mathcal{UL}_2, \mathcal{CL}_2, \mathcal{L}_2^0, X_2, V_2, \mathcal{I}_2, E_2)$ *be two extended timed automata. Their* parallel composition $A_1 \parallel A_2$ *is defined as the timed automaton*

$$A = (\Sigma_1 \cup \Sigma_2, \mathcal{L}_1 \times \mathcal{L}_2, \mathcal{L}_1^0 \times \mathcal{L}_2^0, X_1 \cup X_2, V_1 \cup V_2, \mathcal{I}, E)$$

[6] An exclamation mark is used in case of the send signal. Exclamation marks, resp. question marks allow to graphically distinguishing between send and received signals.

[7] Committed locations become more relevant in the context of the parallel product of two timed automata, where each can or cannot be in a committed location.

where $\mathcal{I}(\langle l_1, l_2\rangle) = \mathcal{I}_1(l_1) \cup \mathcal{I}_2(l_2)$ *for all* $l_1 \in \mathcal{L}_1$, $l_2 \in \mathcal{L}_2$, *and* $(\langle l_1, l_2\rangle, a, \varphi, \lambda, \langle l_1', l_2'\rangle) \in E$ *if one of the following conditions holds:*

1. $a \in \Sigma_1 \setminus \Sigma_2$, $l_2 = l_2'$ *and there exists* $(l_1, a, \varphi, \lambda, l_1') \in E_1$ *and* $l_1 \notin \mathcal{CL}_1, l_2 \notin \mathcal{CL}_2 \vee l_1 \in \mathcal{CL}_1$,
2. $a \in \Sigma_2 \setminus \Sigma_1$, $l_1 = l_1'$ *and there exists* $(l_2, a, \varphi, \lambda, l_2') \in E_2$ *and* $l_1 \notin \mathcal{CL}_1, l_2 \notin \mathcal{CL}_2 \vee l_2 \in \mathcal{CL}_2$,
3. $a \neq \sigma_\epsilon \wedge a \in \Sigma_1 \cap \Sigma_2$, *and there exist* $(l_1, a, \varphi_1, \lambda_1, l_1') \in E_1$, $(l_2, a, \varphi_2, \lambda_2, l_2') \in E_2$ *with* $\varphi = \varphi_1 \cup \varphi_2$, $\lambda = \lambda_1 \cup \lambda_2$.

Further holds each $l_i \in \mathcal{L}_1 \times \mathcal{L}_2$ *with* $l_i =< l_j, l_k >$ *and* $l_j \in L_1$, $l_k \in L_2$ *is an urgent location, if* l_j *or* l_k *is an urgent location and both are not committed locations. Further holds* l_i *is a committed location if* l_j *or* l_k *is a committed location (committed locations dominate urgent locations under composition).*

The semantics of a (extended) timed automaton is defined in terms of an induced timed system.

Definition 5 (Timed system)
A timed system *is a tuple* $\mathcal{T} = (\Sigma, \mathcal{S}, \mathcal{S}^0, T)$ *consisting of a set of signals* Σ *with* $\Sigma \cap \mathbb{R}^+ = \emptyset$, *a set of states* $\mathcal{S}$, *a set of initial states* $\mathcal{S}^0 \subseteq \mathcal{S}$ *and a transition relation* $T \subseteq \mathcal{S} \times (\Sigma \cup \mathbb{R}^+) \times \mathcal{S}$.

A (extended) timed automaton $A = (\Sigma, \mathcal{L}, \mathcal{UL}, \mathcal{CL}, \mathcal{L}^0, X, V, \mathcal{I}, E)$ induces the timed system $\mathcal{T}_A = (\Sigma, \mathcal{S}, \mathcal{S}^0, T)$ in the following way. A function $v : X \to \mathbb{R}^+$ is called a *clock valuation*. For a duration $\tau \in \mathbb{R}^+$, $v + \tau$ is the clock valuation v'' with $v''(x) = v(x) + \tau$ for all $x \in X$. Now, the set of states $\mathcal{S}$ of the induced timed system $\mathcal{T}_A$ is given by all pairs $s = \langle l, v\rangle$ where $l \in \mathcal{L}$ and v is a clock valuation for X as well as an attribute assignment for V. The set of initial states $\mathcal{S}^0$ consists of all pairs $s^0 = \langle l^0, v^0\rangle$ where $l^0 \in \mathcal{L}^0$ and v^0 is the clock valuation that assigns 0 to all $x \in X$ and 0 to all variables of V. The induced timed system $\mathcal{T}_A$ contains two types of transitions:

1. *discrete transitions* $s \xrightarrow{\sigma} s'$ where $\sigma \in \Sigma$ is a signal,
2. *time transitions* $s \xrightarrow{\tau} s'$ where $\tau \in \mathbb{R}^+$ is a duration.

Discrete transitions change the location parameter of a state and are induced by the edges of the TA. A subset of the clocks may be reset to 0 when taking the transition, as specified by the edge. Further, when taking an edge a potentially empty set of variables included in V is updated.

A discrete transition can only be taken in state s if an edge $e = (l, a, \varphi, \lambda, l')$ of the underlying TA is enabled. An edge e of a TA is called enabled in state s, if the TA in state s is in a location l where an outgoing edge e, including the signal $\sigma = a$ (including the empty signal), can be taken. This is only the case if the guard φ can be evaluated to true in state s.

Not reset clocks and not updated attributes remain the same. Time transitions do not modify the location, but increase the values of all clocks synchronously, i.e. $\langle l, v\rangle \xrightarrow{\tau} \langle l, v+\tau\rangle$. This is possible only as long as the invariant of location l is satisfied and if l is not an urgent or committed location. Since the induced timed system of a TA is usually infinite, in practice symbolic representations of the clock valuations, such as *clock zones* [14], are used to obtain a finite semantics.

The induced timed system $\mathcal{T}_{A_1 \| A_2}$ of the parallel product $A_1 \parallel A_2$ of the two TA $A_1 = (\Sigma_1, \mathcal{L}_1, \mathcal{UL}_1, \mathcal{CL}_1, \mathcal{L}_1^0, X_1, V_1, \mathcal{I}_1, E_1)$ and $A_2 = (\Sigma_2, \mathcal{L}_2, \mathcal{UL}_2, \mathcal{CL}_2, \mathcal{L}_2^0, X_2, V_2, \mathcal{I}_2, E_2)$ accordingly contains states of the form $s = \langle l, v\rangle$ with $l = \langle l_1, l_2\rangle$ and $l_1 \in \mathcal{L}_1$, $l_2 \in \mathcal{L}_2$ and v being an assignment of all clocks included in the set $X_1 \cup X_2$ as well as variables in $V_1 \cup V_2$. Discrete transitions accordingly include signals $\sigma \in \Sigma_1 \cup \Sigma_2$.

The timed system of a given TA A is denoted by $\mathcal{T}_A = (\Sigma_A, \mathcal{S}_A, \mathcal{S}_A^0, T_A)$. $T_A(s_A)$ denotes the set of transitions starting in state $s_A \in \mathcal{S}_A$ of the induced timed system of TA A.

The observable behavior of a timed automaton A is defined by the set of traces observable on $\mathcal{T}_A$. A trace α is a finite or infinite alternating sequence of signals and durations $\alpha = \tau_1\sigma_1\tau_2\sigma_2\tau_3\ldots$ with $\sigma_i \in \Sigma$ and $\tau_i \in \mathbb{R}^+$ specifying the order and the delays between consecutive signals. $signals(\alpha)$ denotes the set of signals that occur in α. Furthermore, let $traces\_to(s_1, A)$ denote all traces starting in an initial state and ending in state s_1 and $traces\_from\_to(s_1, s_2, A)$ denotes the set of all observable traces of the TA A when starting in state s_1 and ending in state s_2. Let $traces\_from(s, A)$ denote the set of all observable traces of a timed automaton A starting in state s and let $traces(A)$ denote the union of all traces observable from all initial states. If a set of traces is given explicitly, it is also referred as a *language* and usually denoted by the letter Υ. $signals(\Upsilon)$ denotes the set of signals included in at least one trace $\alpha \in \Upsilon$. Attributes of a TA are defined to be internal and are not shared with other TA in a parallel product. As a consequence attributes do not occur in any trace or language of a TA.

With $time(\alpha)$ the sum over all time delays included in α is denoted. For trace $\alpha = 1.13\, a\, 2.27\, b\, 0$ it holds that $time(\alpha) = 3.4$.

Definition 6 (Time Length)
Let t be either a transition $t = s \xrightarrow{\tau} s_1'$ observable on a state s of the induced timed system of a TIOA or let $t = \tau_1\sigma_1\tau_2\sigma_2\ldots\tau_n\sigma_n$ be a trace. The function time-length $time(t)$ over t is defined as follows:

- *In case t is a transition, $time(t) = 0$ if t is a discrete transition and $time(t) = \tau$ if t is a continuous transition.*
- *In case t is a trace $t = \tau_1\sigma_1\tau_2\sigma_2\ldots\tau_n\sigma_n$ with τ_i being a delay and σ_i being a signal for $0 \leq i \leq n$, $time(t)$ is equal to a such that $\sum_{i=1}^{n} \tau_i = a$.*

Further, the language of a TA is defined as follows.

Definition 7 (TA Language)
The language Υ of a (extended) TA A is defined as the set of traces $\Upsilon = traces(A)$. $traces(A)$ includes all traces that can be observed on the TA A starting from an initial state.

For being able to distinguish between different sets of signals included in a TA at the level of its language, the restriction of a language is following defined.

Definition 8 (Language Restriction)
Let α be a trace and σ be a signal. The trace $\alpha \setminus \sigma$ is derived from α by replacing all occurrences of $\tau\sigma\tau'$ in α by $(\tau+\tau')$ for all $\tau, \tau' \in \mathbb{R}^+$. Let Σ be a set of signals and $\Sigma_\Upsilon = \{\overline{\sigma_1}, \ldots, \overline{\sigma_n}\} = signals(\Upsilon) \setminus \Sigma$ all signals included in Υ but not in Σ. The trace $\alpha{\restriction}\Sigma$ is defined as $\alpha{\restriction}\Sigma = \alpha \setminus \overline{\sigma_1} \setminus \ldots \setminus \overline{\sigma_n}$. For a given language Υ, $\Upsilon{\restriction}\Sigma$ is equal to $\{\ \alpha{\restriction}\Sigma \mid \alpha \in \Upsilon\ \}$.

Thus, for a language Υ, the language $\Upsilon{\restriction}\Sigma$ is derived by restricting the set of included signals to Σ, formally by removing all signals that do not occur in Σ in all traces of Υ. Note that when removing the occurrence of a signal in a trace, the delays before and after this occurrence are summed up in the derived trace.

Moreover, a trace α is called a *prefix* of a trace β, written as $\alpha \vdash \beta$, if $\alpha = \beta$ or α is a finite trace $\alpha = \tau_1\sigma_1 \ldots \tau_n\sigma_n$ and β is of the form $\beta = \tau_1\sigma_1 \ldots \tau_n\sigma_n\tau_{(n+1)}\sigma_{(n+1)}\ldots$, i.e., β begins with α. Further, in a state s of an induced timed system an observable transition t is part of the observable trace α starting from this state. $t \vdash \alpha$ denotes that t contributes the first step of the observable trace α starting from state s. Also two traces t_1 and t_2 can be related by the operator $\vdash$. As an example consider $t_1 = s_1 \xrightarrow{a} s_1'$ and $t_2 = s_2 \xrightarrow{b} s_2'$. If a is a signal,$t_1 \vdash t_2$ is fulfilled if $a = b$. If a is time delay greater than zero, $t_1 \vdash t_2$ is fulfilled if b is time delay with $a \leq b$.

$\alpha = \beta \circ \gamma$ denotes the concatenation of two traces and $\alpha = \beta \circ t \circ \gamma$ denotes the concatenation of a trace β, a transition t and a trace γ.

The restriction operator is also defined for a TA.

Definition 9 (TA Restriction)
Let $A = (\Sigma, \mathcal{L}, \mathcal{L}^0, X, \mathcal{I}, E)$ be a TA and let Σ_{rem} be a set of signals. $A{\restriction}\Sigma_{rem} = A_2 = (\Sigma_{rem}, \mathcal{L}, \mathcal{L}^0, X, \mathcal{I}, E_2)$. Let $\Sigma_2 = \Sigma \setminus \Sigma_{rem}$ be the set of removed signals. For each edge $e \in E$, with $e = (l, a, \varphi, \lambda, l')$, holds, $e \in E_2$ if $a \notin \Sigma_2$. If $a \in \Sigma_2$ an edge $e_{rem} = (l, \sigma_\epsilon, \varphi, \lambda, l')$ is included in E_2 containing the empty signal σ_ϵ. Further holds there not exists an edge e in E_2 if one of the following conditions holds:

1) $e = (l, a, \varphi, \lambda, l')$ and $a \in \Sigma_2$ with $e \in E$, or,

2) $e = (l, \sigma_\epsilon, \varphi, \lambda, l') \wedge \neg\exists e_2 : e_2 \in E$ with $e_2 = (l, a_2, \varphi, \lambda, l') \wedge a_2 \in \Sigma_2$ and $e \notin E$.

Informally a TA restricted to a set of signals Σ_{rem} is a TA where signals are removed from edges of the TA that are not included in Σ_{rem} (edges are preserved but signals are added).

Lemma 2.1 (Restriction Relation)
Given a TA A with $traces(A) = \Upsilon$. For any set of signals Σ_{rem} holds: $traces((A{\restriction}\Sigma_{rem})) = \Upsilon{\restriction}\Sigma_{rem}$.

Proof 2.1
(sketch) Assuming $traces((A{\restriction}\Sigma_{rem})) \neq \Upsilon{\restriction}\Sigma_{rem}$. This can only be the case if 1) a trace α is included in $traces((A{\restriction}\Sigma_{rem}))$ that is not included in $\Upsilon{\restriction}\Sigma_{rem}$, or, 2) a trace α is not included

in $traces((A{\restriction}\Sigma_{rem}))$ that is included in $\Upsilon{\restriction}\Sigma_{rem}$. Assuming 1): This can only be the case if a trace α is observable on the induced time system of $A{\restriction}\Sigma_{rem}$ for that holds: $\neg\exists\beta \in \Upsilon$ such that $\beta{\restriction}\Sigma_{rem}$ being equal to α. If this is the case, the restriction operator ${\restriction}\Sigma_{rem}$ applied on A has enabled a transition in the induced time system of $A{\restriction}\Sigma_{rem}$ by removing a signal. Because A is an isolated TA which do not need to synchronize with other TA at all, this cannot be the case. Assuming 2): This can only be the case if a signal σ, included in an edge of A is removed by applying the restriction operator ${\restriction}\Sigma_{rem}$ on A, such that for at least one trace β, which is observable on the induced time system of A, no trace α is observable on the induced time system of $A{\restriction}\Sigma_{rem}$ with $\beta{\restriction}\Sigma_{rem}$ being equal to α. This can only be the case if a state of the induced time system of A is reachable because a transition is enabled only if synchronization with σ appears. Because in our model such synchronization of an isolated TA can only disable but not enable transitions, this cannot be the case.

Informally Theorem 2.1 states that an equivalence relation between the observable traces of a TA and the language of the TA concerning the restriction operator exists. Thus, it doesn't matter if first the TA is restricted to a set of signals or if the set of observable traces, of the induced timed system, is restricted to the same set of signals. Both methods lead to the same set of signals.

2.2 Timed I/O Automata

The definition of a timed I/O automaton (TIOA) is obtained by explicitly distinguishing between input, output and internal signals contained in a timed automaton.

Definition 10 (Timed I/O automaton)
A timed I/O automaton *(TIOA) is a tuple $\mathcal{A} = (A, I, O, H)$ where A is a (extended) timed automaton and the set of signals Σ of A is partitioned into the disjoint sets I, O and H, respectively called the* input, output *and* internal *signals of $\mathcal{A}$. Further holds that the empty signal is an internal signal: $\sigma_\epsilon \in H$.*

Intuitively, a component modeled by a TIOA receives input signals, sends output signals and performs internal state changes using internal signals or delay transitions.

Definition 11 (Parallel composition of TIOA)
Given two timed I/O automata $\mathcal{A}_1 = (A_1, I_1, O_1, H_1)$ and $\mathcal{A}_2 = (A_2, I_2, O_2, H_2)$. Their parallel composition is defined as $\mathcal{A}_1 \parallel \mathcal{A}_2 = (A_1 \parallel A_2, I, O, H)$ where:

- $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$,
- $O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$,
- $H = H_1 \cup H_2 \cup (I_1 \cap O_2) \cup (I_2 \cap O_1)$

if the following conditions hold:

1. $I_1 \cap I_2 = O_1 \cap O_2 = \emptyset$
2. $H_1 \cap (I_2 \cup O_2 \cup H_2) = H_2 \cap (I_1 \cup O_1 \cup H_1) = \emptyset$,
3. $X_1 \cap X_2 = \emptyset$ *where* X_1, X_2 *are the clock sets of* A_1 *and* A_2*, respectively.*
4. $V_1 \cap V_2 = \emptyset$ *where* V_1, V_2 *are attribute variable sets of* A_1 *and* A_2*, respectively.*

Thus, two TIOA can be composed only if (1) their respective input and output signal sets are disjoint, (2) the internal signals of one automaton do not occur in the other automaton, (3) they do not share any clocks and (4) attributes. If these conditions are fulfilled the two TIOA are following called *compatible*. When modeling components using TIOA, the parallel composition of two components is only allowed if the associated TIOA holds: input ports are connected to output ports and vice versa. Internal signals sets need to be disjoint and variables (clocks and attributes) of different components, resp. TIOA also need to be disjoint. Moreover, connected ports become internal. In this way, parallel composition can be used to hierarchically structure a component-based system (see Sec. 6.2).

In this paper the focus is set on undesired behavioral properties of TIOA in the form of deadlocks and zeno-behavior. Formally, for a TIOA $\mathcal{A}$, a finite trace α_d is called a *deadlock trace* if there exists a path for α_d in $\mathcal{T}_A$, the induced timed system of $\mathcal{A}$, that ends in a state s_d that contains no transition. A deadlock state is defined accordingly.

Definition 12 (Deadlock)
Let $\mathcal{T}_A = (\Sigma, \mathcal{S}, \mathcal{S}^0, T)$ *be the induced timed system of a TIOA* $\mathcal{A}$*. A state* $s_d \in \mathcal{S}$ *is a deadlock state if* $\neg\exists t \in T$ *with* $t = s_d \xrightarrow{\sigma} s'_d$ *and* $\sigma \in \Sigma$ *or* $t = s_d \xrightarrow{\tau} s'_d$ *and* $\tau > 0 \wedge \tau \in \mathbb{R}^+$*.*

Informally a deadlock state is a state where no outgoing transition can be taken including a signal or a delay greater than zero.

An infinite trace $\alpha_z = \tau_1\sigma_1\tau_2\sigma_2\ldots$ is called a *zeno trace* if there exists an $\ell \in \mathbb{R}^+$ such that $\sum_{i=1}^{\infty} \tau_i = \ell$, i.e., if it has a finite time length. In contrast to a *deadlock*, for a *zeno trace* no associated state exists. As a consequence no definition of a zeno state is given.

3 Application Example - Engine Control

Now that required preliminary formal definitions for TA and TIOA have been given, the application example of the engine-control model shown in Fig. 1 is described in more detail. Like previously mentioned this application example is derived from an existing demo application shipped with the professional tool SystemDesk.[8] We first give a more detailed description of each included component. Afterwards we specify the timing behavior for each included component by an individual TIOA. This example is used in the remainder of this work to illustrate how the framework, which is introduced in the next sections, can be applied on typical applications in the domain of real-time embedded systems.

[8] See http://www.dspace.com/systemdesk

3.1 Component Engine-Model

The component *EngineModel* is an abstract representation of the behavior of the physical engine. It has two main functionalities: first, consuming the desired fuel rate and second, calculating appropriate raw output values. Within the raw output, information about the current throttle position, the actual speed value of the engine, the oxygen value as well as the actual pressure (e.g., of the combustion) is included. Because this information is always sent at the same point of time, only a single signal *raw* is considered within the subsequently introduced TIOA representing the timing-behavior of the component. Because in this work the focus is set on the non-functional and especially the timing-behavior, concrete values of send or received signals are following not considered.

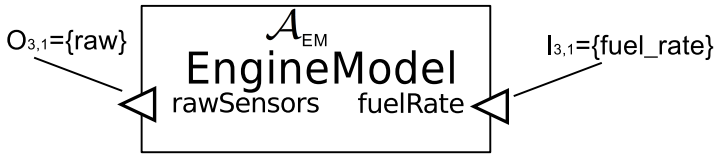


Figure 2: The component representing the physical behavior of the engine.

The associated TIOA is shown in Fig. 3. It consists of four locations. In the initial location *Init* the component is able receiving the current value of the fuel rate via the signal *fuel_rate*. In the second location the TIOA waits for executing functionality and the third location (*CRV* as a shortcut for Calculate-Raw-Values) is associated with a state, where values received via the signal *fuel_rate* are send to the real engine[9] and raw output values are read inside the component.[10] At this location the component is not able receiving arriving signals of type *fuel_rate*. This is the case because in location *CRV* consistent values need to be send to the real engine. The fourth location *wait* is associated with the state where the component has finished its computation and is waiting for the next period to begin.

Following, the TIOA representing the timing behavior of component *EngineModel* is described in more detail. For graphically describing the structure of a TA, the same syntax like provided by the model checker UPPAAL (cf. [8]) is used. Each input signal is followed by a question mark and each output signal is followed by an exclamation mark. In such a manner, in the remainder of this work, input and output signals of a TIOA can be distinguished. The TIOA is allowed to stay in the initial location for at most five time units, according to the invariant of the form $p3 \leq 5$, where $p3$ is a clock for measuring the progress of time in each period. Thus, location *Init* needs to be left at point in time five. The only outgoing edge need to receive the signal *fuel_rate*. Thus, in the initial location the TIOA need to receive this signal within a time frame of five milliseconds (ms). After five ms the TIOA is required to leave the second location taking the edge to location *CRV* (Calculate Raw Values). When taking the edge to location *CRV*, clock $d3$ is reset to zero. The clock is used for measuring the amount of time spent for writing respectively reading sensor and actuator values when being in location *CRV*. The invariant $d3 \leq 1$ requires the TIOA leaving the location *CRV* at

[9] This is realized by setting actuators of the real engine.

[10] By reading and processing the output of analog-digital converters of the sensors of the real engine.

the latest after one time unit and the guard allows to leave the location also at the earliest after one time unit. When leaving location *CRV*, signal *raw* is send via the edge leading to location *wait*. The TIOA is required to stay in this location till the overall period of 10 time units is over (realized by the guard $p3 \geq 10$ in combination with the invariant $p3 \leq 10$). Afterwards, the edge to the initial location is taken, resetting the clock $p3$ to zero. The resulting TIOA $\mathcal{A}_{EM} = (A_{EM}, I_{EM}, O_{EM}, H_{EM})$ consists of the input signal $I_{EM} = \{fuel\_rate\}$, the output signal $O_{EM} = \{raw\}$, an empty set of internal signals $H_{EM} = \emptyset$ and the TA A_{EM} with constituent elements like described above.

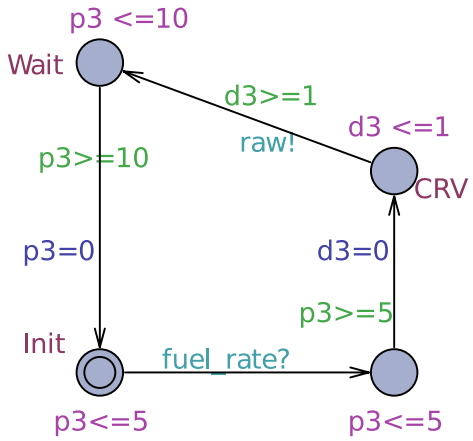


Figure 3: The TIOA $\mathcal{A}_{EM}$ representing the engine model.

3.2 Component Fuel-Sensor

The component *FuelSensor* is responsible for evaluating the raw values provided by the component *EngineModel*. Depending on the current values and the absolute point of time in which they have been received, the mode of the engine is calculated and sent via a signal to the component *FuelController*. In case the engine is running for a short period (still in the warm up phase) or a failure is detected, the mode is set to *rich*. In this mode the component *FuelController* is responsible for calculating a fuel rate allowing the engine to run in a more robust way. In the case the engine is already warm and no failure is detected the mode is set to *normal*, requiring the component *FuelController* to calculate more optimized values for the fuel rate. Further, *FuelSensor* is responsible for realizing a post-processing of the raw values, allowing component *FuelController* more easily evaluating the raw value (e.g., removing sensor-failures if possible), resulting in the output signal *eng_value*.

The associated TIOA is shown in Fig. 5. It contains the initial location *Init*. When being in this location, signals of type *raw* can be received by the component. In the successor location *DSF* (short for Detect-Sensor-Failures) the mode is derived. Depending on the absolute point in time and the received values the mode is set either to *normal* or to *rich* by sending the signal *normal*, resp. *rich*. When sending the signal *rich* or *normal*, the location with name *RSC*, respectively *RSC2* (Run-Sensor-Correction) is reached. The only difference between

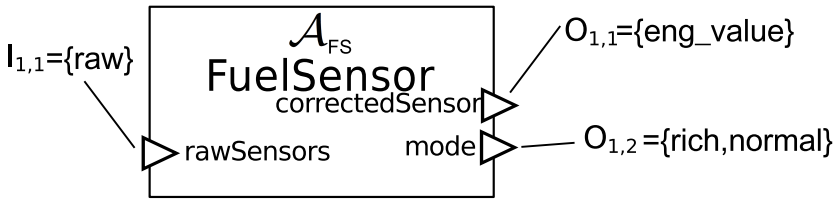


Figure 4: The component representing functionality for processing raw sensor values.

these two locations is, that in case of being in mode *rich* a more simplified computation is used for processing the input values received via the signal *raw*. Accordingly, varying execution times are associated with both locations (RSC and RSC2). Afterwards the signal *eng_value* is sent via the edge leading to location *wait*, where the component is waiting for the end of the period before taking the edge to the initial location. In the initial location, as well as in location *wait*, the component is able to receive incoming signals of type *raw*. In the remaining locations this is not allowed due to the fact that in locations *DSF* and *RSC* consistent data need to be provided when computing output signals.

Concerning the timing the associated TIOA $\mathcal{A}_{FS}$ behaves as follows. $\mathcal{A}_{FS}$ is allowed to stay in the initial location for at most seven time units reflected by the invariant $p \leq 7$, where p is a clock for measuring the current time of the overall period. When taking the edge to location *DSF* (Detect-Sensor-Failures) the clock d is reset to zero. Clock d is used for measuring the duration of the computation of each location (for locations associated with computations like described above). The computation of the error-detection takes at most one time unit reflected by the invariant $d \leq 1$ in combination with the guard $d \geq 0$. In addition the outgoing edge to location *RSC* (Run-Sensor-Correction) has the guard $abs \geq 1000$ for ensuring that the signal *normal* is only send if the engine is running for a longer time period. Depending on the current mode the successor location *RSC* resp. *RSC2* is reached. In comparison to location *RSC2*, where a more simplified processing of the raw values is applied, location *RSC* requires more computation time (between $1-2$ ms in contrast to $0-1$ ms). The guards, invariants and resets, like shown in Fig. 5, realize this timing behavior. When leaving location *RSC*, resp. *RSC2*, the signal *eng_value* is sent via the edge leading to location *wait*. At this location the TIOA is waiting till the overall period, measured by the clock p, is over.

The resulting TIOA $\mathcal{A}_{FS} = (A_{FS}, I_{FS}, O_{FS}, H_{FS})$ consists of the input signal $I_{FS} = \{raw\}$, the output signals $O_{FS} = \{rich, normal, eng\_value\}$, an empty set of internal signals $H_{FS} = \emptyset$ and the TA A_{FS} with constituent elements like described above. Zeno behavior in case of the incoming signals is prevented like previously described in the case of the TIOA representing the component *EngineModel*, using additional attribute variables. The complete resulting TIOA can be found in Appendix B.

Some of the components of the real application example, taken from the professional tool SystemDesk, allow receiving an arbitrary number of signals when not being in a state where signals are processed. This is also the case for component *FuelSensor* that is able to receive an arbitrary number of signals of type *raw* when being in location *Init*. This is the case because the last received value is considered to be the best value (last-value best-value

semantic). Unfortunately, allowing receiving an arbitrary number of signals introduces an undesired property of TIOA in form of zeno behavior. Also for any real implementation it cannot be considered that a component is able to receive an unbounded number of incoming signals in finite time. To avoid undesired behavior in case of zeno traces and to achieve a more realistic component behavior, it is only allowed to receive a limited number of signals when being in a location. For this purpose a dedicated single attribute variable *a* is used within the TIOA shown in Fig. 5. Each edge that receives signal *raw* has an additional guard ensuring that at most a maximal number of signals can be received while being in locations *Init*. Each time a signal *raw* is received the attribute variable is incremented by one. Only in case an edge is taken from a location not being able to receive signals to location *Init*, this attribute variable is reset to the value zero. For keeping the TIOA models simple and for allowing a better understanding, these additional variables and guards are not shown in Fig. 29. The complete TIOA models, including variables, guards and updates avoiding zeno behavior, can be found in Appendix B.

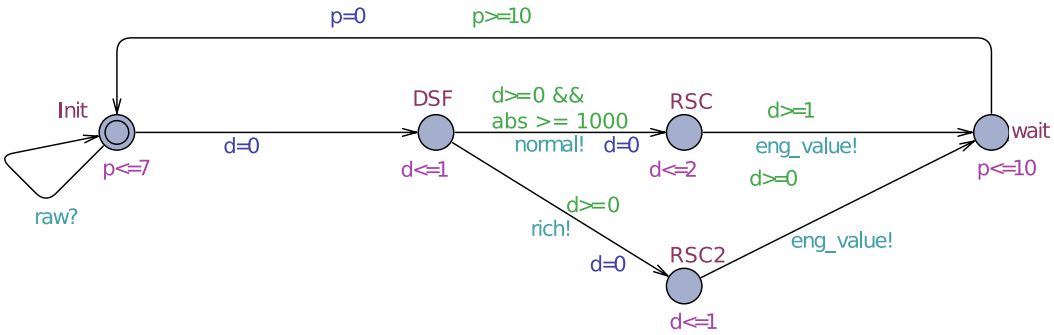


Figure 5: The TIOA $\mathcal{A}_{FS}$ representing the timing behavior of component *FuelSensor*.

3.3 Component Fuel-Controller

The component *FuelController* is responsible for calculating the appropriate fuel rate for the engine. Depending on the current mode as well as the current sensor values different computations are executed. In every case the approximate current airflow is calculated based on the input values received via the port *correctedSensor*. Depending on the current mode and the approximated airflow value the desired fuel rate is calculated, either for a more robust behavior when being in the mode *rich* or for a more optimized behavior when being in mode *normal*. In our model a mode-switch is indicated each time an appropriate signal is received via the port *mode*. Depending on the mode different functional parts are executed. When being in mode *rich*, a fuel rate appropriate for a more robust but less efficient engine operation is calculated. When being in mode *normal*, more optimized values for the fuel rate are derived.

Fig. 7 shows an example how the state-based timing-behavior of the *FuelController* can be modeled using a TIOA. The TIOA basically describes two different behaviors, represented by the locations and edges at the upper respectively lower part of Fig. 7. Those located at the

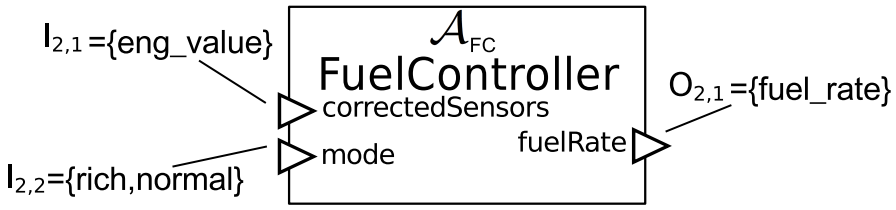


Figure 6: The component representing the Fuel-Controller.

lower part do represent the behavior of the *FuelController* when being in mode *rich*. Those located at the upper part represent the behavior when being in mode *normal*. In the locations on the left as well as the locations on the right the component is able to receive incoming signals in case of signals *eng_value*, *rich* and *normal*. When receiving signal *rich* or *normal*, the TIOA switches accordingly between the upper and lower locations, representing the different modes. In the upper part of the TIOA the location *NAC* (Normal-Airflow-Calculation) represents the state where the component *FuelController* calculates the airflow when being in mode *normal*.[11] In this mode, in contrast to mode *rich*, additional functionality for deriving a corrected and accurate value for the airflow is executed. This airflow is required for the successor location *NFRC* (Normal-Fuel-Rate-Calculation) where the optimized desired fuel rate is calculated. At the lower part of the TIOA shown in Fig. 7, locations and edges are shown representing the behavior parts for calculating the fuel rate when being in mode *rich*. In contrast to location *NAC*, location *RAC* (Rich-Airflow-Calculation) computes the resulting airflow without any correction. In location *RFRC* (Rich-Fuel-Rate-Calculation) a fuel rate is calculated allowing operating the engine in a more robust way.

The timing of the two different behavior parts, associated with the mode *rich* respectively *normal* like shown in Fig. 7, does only differ in case of locations *NAC* and *RAC*. Location *NAC*, associated with the normal execution mode can require more computation time for calculating the airflow compared to location *RAC*. The rest of the behavior is almost identical, starting in one of the locations on the left. Invariants require leaving these locations within the first two time units. The clock $d2$ is set to zero when taken the edge to location *NAC*, resp. *RAC*. The clock is used to reflect the execution times like described above by using appropriate guards and invariants. The timing behavior of the two different successor locations *NFRC* and *RFRC* is identical, again realized by an invariant ($d2 \leq 1$), indicating that the computation of the resulting output signal *eng_value* is finished within one time unit. In the locations on the right the clock $p2$ is used to ensure that the TIOA will only switch back into the initial location when the overall period (10 time units) is over.

The resulting TIOA $\mathcal{A}_{FC} = (A_{FC}, I_{FC}, O_{FC}, H_{FC})$ consists of the input signals $I_{FC} = \{rich, normal, eng\_value\}$, the output signal $O_{FC} = \{fuel\_rate\}$, an empty set of internal signals $H_{FC} = \emptyset$ and the TA A_{FC} with constituent elements like described above. Again, for avoiding zeno behavior an additional attribute variable is used. The resulting complete TIOA can be found in Appendix B.

[11] The airflow is not considered within the example. The airflow can be considered as an internal signal that is hidden for allowing a better understanding.

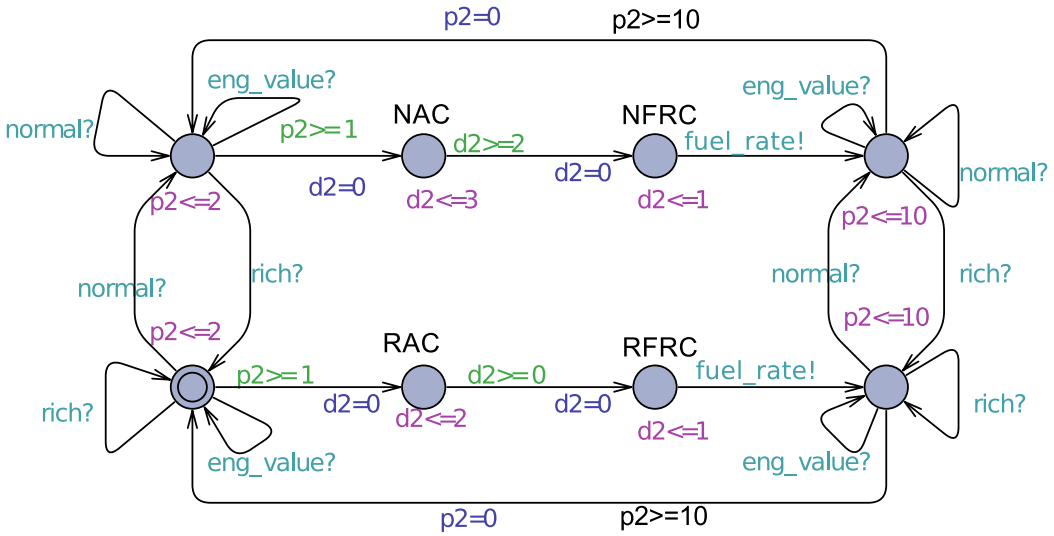


Figure 7: TIOA $\mathcal{A}_{FC}$ representing the timing behavior of component *Fuel-Controller*.

The introduced application example is rather small compared to real applications in the domain of embedded real-time systems, containing a huge number of components. Nevertheless, also in case of the introduced example major differences concerning the complexity of the analysis are shown in Sec. 8.3 when applying the subsequently introduced scalable reasoning scheme, compared to a monolithic approach where all involved TIOA are analyzed at once. Further, it can be seen that the application example, as a typical example for real-time embedded systems, provides components with restricted resources (R2), not being able to receive any amount of signals at any state. As an example see the TIOA shown in Fig. 7, not being able receiving signals when being in one of the locations *NAC*, *NFRC*, *RAC* and *RFRC*. Further, also in the case of this more simple application example non-deterministic behavior exists. This is the case because execution times of functional parts can only be expressed using upper and lower bound (associated with Best-Case and Worst-Case execution times) in form of invariants and guards of the associated TIOA. As a consequence, like in the case of the TIOA shown in Fig. 7, state-changes occur nondeterministically. As an example the transition between location *NAC* and *NFRC* cannot be predicted based on the observation of time and exchanged signals only. As a consequence, approaches that do require deterministic behavior for realizing a modular and scalable check (cf. [11]) seem to be too restrictive to be applied on such application examples. While the shown behavior models have been derived manually from a given architecture model,[12] such TA models can also be automatically derived like shown in [19].

[12] In case of a given AUTOSAR model.

4 Progressive and Receptive TIOA

Following the notion of progressive and receptive behavior is introduced, like defined in [16]. It is shown that progressive and receptive behavior requires strong assumptions that are hard or even impossible to achieve for resource restricted embedded components. Based on the given property of progressive behavior a new property is defined as a relaxed condition of progressive and receptive behavior that does not require such strong assumptions. This relaxed property is used to verify the correctness of the previously introduced example.

In [16] desired properties, like in the case of the absence of deadlock and zeno traces, is captured by the notion of *progressive* TIOA. A TIOA $\mathcal{A} = (A, I, O, H)$ is called *progressive* if the induced timed system does not allow to observe any zeno executions or deadlocks if the environment includes no deadlocks and zeno behavior. The definition of progressive behavior according to [16] is rather strong. Fulfilling the property of progressive behavior, according to [16], a TIOA is also I/O feasible, what means that in each state of $\mathcal{A}$ an arbitrary trace $\beta = \tau_1, \sigma_1, \tau_2, ...$, consisting of signals $\sigma_i \in I$ and of arbitrary delays τ_i, can be accommodated by $\mathcal{A}$. Thus, an I/O feasible TIOA automaton $\mathcal{A} := (A, I, O, H)$ is able to accommodate arbitrary input actions (signals) occurring at arbitrary time. For an I/O feasible TIOA $\mathcal{A}$ holds that each trace $\alpha := \tau_1\sigma_1\tau_1\sigma_1...$, consisting of signals $\sigma_i \in I$, $\alpha \in traces\_from(\omega, \mathcal{A})$ can be observed on $\mathcal{A}$ starting in any state s of $\mathcal{A}$, respectively of the induced timed system.

However, the requirement of progressiveness of components is rather strong. According to [16] instead of progressive behavior also receptive behavior, as a weaker condition, can be used. Receptive behavior is based on the definition of a strategy. Therefore, first it is defined what a strategy is in the context of a TIOA. For being able to reason about a strategy the edges of a TA, and as result also of a TIOA are partitioned into *controllable* and *uncontrollable* once.

Definition 13 (Disjoint Partitioned TA)
A Disjoint Partitioned TA *is defined as follows. Let* $A = (\Sigma, \mathcal{L}, \mathcal{UL}, \mathcal{CL}, \mathcal{L}^0, X, V, \mathcal{I}, E)$ *be a (extended) TA where a disjoint partitioning of controllable edges* E_n *and uncontrollable edges* E_c *with* $E_n \cap E_c = \emptyset$ *and* $E_n \cap E_c = E$ *exists.* $A' = (\Sigma, \mathcal{L}, \mathcal{UL}, \mathcal{CL}, \mathcal{L}^0, X, V, \mathcal{I}, E')$ *containing uncontrollable edges* $E_n \in E$ *with* $E_n \in E'$ *and controllable edges* $E'_c \cap E_n = \emptyset$ *with* $E'_c \cup E_n = E'$ *is a* Disjoint Partitioned TA *of* $\mathcal{A}$*.*

Based on the definition of a partitioning of the edges of a TA, now the semantics of a valid strategy is defined. In the reminder of this work, whenever a strategy is mentioned a valid strategy according to Def. 14 is meant.

Definition 14 (Valid Strategy)
Given TA $A = (\Sigma, \mathcal{L}, \mathcal{UL}, \mathcal{CL}, \mathcal{L}^0, X, V, \mathcal{I}, E)$*. Let* $A' = (\Sigma, \mathcal{L}, \mathcal{UL}, \mathcal{CL}, \mathcal{L}^0, X, V, \mathcal{I}, E')$ *be a* Disjoint Partitioned TA A' *of* A *according to Def. 13, where* $E = E_c \cup E_n$ *and* $E' = E'_c \cup E'_n$*.* A' *is a valid strategy for* A *if the following conditions are fulfilled: 1)* $E_n = E'_n$*. 2)* $\forall e = (l_1, a, \varphi, \lambda, l'_1) \in E'_c$ *holds:* $\exists e_1 = (l_1, a, \varphi', \lambda, l'_1) \in E_c$ *and for all states* s *of the induced time system* $\mathcal{T}_{A'}$ *of* A'*, being in location* l *holds: if* φ' *is enabled also* φ *is enabled (*$\varphi' \implies \varphi$*). Further, for a valid strategy holds that no deadlock state or zeno trace is observable on* A'*.*

A strategy is valid if and only if controllable edges are removed or modified in such a way that the resulting observable behavior is a subset of the observable behavior of the original TA A. As a consequence each valid strategy A' realizes a subset of the observable behavior of the original TA A, like in the case of the induced timed system and the observable traces. As a consequence holds for each valid strategy A' of a TA A: $traces(A') \subseteq traces(A)$.

The notion of a strategy can be applied to the model of TIOA by simply allowing only edges including signals of the set of internal signals to be controllable. Thus, a valid strategy $\mathcal{A}' = (A', I, H, O)$ of a TIOA $\mathcal{A} = (A, I, H, O)$ is defined based on the TA A included in TIOA $\mathcal{A}$ where an edge of A is only allowed to be controllable if only signals of H are included in this edge. In such a manner different strategies of different TIOA are kept independent from each other under composition. This is the case because only edges including internal signals are allowed to be controllable. Like also done in [16] strategies are used similar to winning strategies as in [5] for different types of games for timed automata. Defining strategies based on timed automata allows avoiding introducing extra mathematical machinery.

The definition of receptive behavior in [16] is based on progressive behavior and a timed I/O automaton $\mathcal{A}$ is called *receptive* if there exists a strategy $\mathcal{A}'$ for $\mathcal{A}$ that is progressive. In other words, a TIOA is receptive if a strategy $\mathcal{A}'$ exists that is able to resolve all nondeterministic choices in such a manner that $\mathcal{A}$ is not able to generate infinitely many internal transitions in finite time, no matter how the environment behaves. Furthermore, for a receptive TIOA it is required that in the case $\mathcal{A}$ is executed according to strategy $\mathcal{A}'$, time can always progress if the environment allows to do so. As a consequence no deadlock and no zeno behavior can occur for the strategy $\mathcal{A}'$ of a receptive TIOA $\mathcal{A}$. For example, if a component in the environment decides to send no signal to $\mathcal{A}$ within the time period t, $\mathcal{A}$ under A' allows to let time pass for at least t time units without receiving any signal.

Finding a strategy and thus determining whether a TIOA is receptive or not can be decided using timed two player games (see [9, 16]). For this purpose, edges of a TIOA need to be partitioned into *controllable* and *uncontrollable* according to Def. 13, depending on whether the observed signal σ is internal, i.e. $\sigma \in H$, or not.

In a timed two player game (cf. [9]) a player can choose controllable transitions and its opponent can choose the uncontrollable transitions. Depending on the winning condition, e.g., reaching or avoiding specific states, such a timed two player game eventually results in a strategy, indicating that a strategy $\mathcal{A}'$ according to Def. 14 can exist, where the player is able to win the game no matter which choices its opponent makes.

Taking two compatible TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$ into consideration while both on their own fulfill all required properties like in the case of progressive or receptive behavior. One reasonable question is if such properties are preserved under composition. In [16] this question has been answered for progressive TIOA.

Theorem 4.1
(from [16]) If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible progressive TIOA, then their composition is also progressive.

Further it has been shown in [16] that the composition of two compatible receptive TIOA

is also receptive. Thus, the property of progressive behavior is sufficient for preserving properties like deadlock freedom and absence of zeno behavior under composition. It is not required to compare the possible input, respectively output traces of the individual automata. Further holds that receptive behavior is preserved under composition, still allowing finding a strategy avoiding zeno and deadlock behavior for the composition.[13] Each progressive or receptive TIOA needs to be also I/O feasible (see [16]). In the following it is shown that even I/O feasibility is hard to achieve for embedded systems and as a result progressive and receptive behavior is even more unrealistic. An I/O feasible automaton $\mathcal{A}$ is at least capable to accommodate any sequence of arbitrary input signals occurring at arbitrary points of time. On the one hand, this property allows ignoring the input traces for the considered components, which are able to consume any sequence of signals when being composed. Therefore, it helps protecting IP (R3) as well as supports a modular, scalable analysis (R1). On the other hand, requiring an automaton to be I/O feasible is an unrealistic or at least very difficult to match property for a relevant portion of typical embedded real-time systems. This is the case because I/O feasibility does not allow specifying a concrete upper bound of signals that a receptive or progressive TIOA $\mathcal{A}$ needs to be able to consume. If the property of I/O feasibility holds, $\mathcal{A}$ can consume any trace α with time length n including m signals. Consider a time length of $n = 10$, $\mathcal{A}$ needs to be able to consume any amount of signals within this 10 time units, e.g., $10, 1000, 10000000$ signals or even more. Only zeno traces, consisting of an infinite number of signals are not allowed. Obviously, such a property can rarely be achieved for a huge portion of resource restricted real-time embedded systems. This is the case because only limited resources can be assumed (requirement (R2)). Only for systems with unrestricted resources or system that are able to simply ignore signals (simply skip incoming signals without consuming resources when skipping signals), the property of receptive or progressive behavior seems to be realistic. As a result the existing TIOA framework can rarely support requirement R2. An example of a TIOA $\mathcal{A}_r$ is shown in Fig. 8. $\mathcal{A}_r$ consists of the tuple (A, I, H, O), with $I = \{a,b\}$, $O = \{d\}$, $H = \{\sigma_\epsilon\}$ and the single clock x. The initial location *READY* has an outgoing edge leading to location *Done*. This edge includes the guard $x \geq 0$ over the clock x. As a result the edge can be taken if the signal a is received and the guard is fulfilled. A second cyclic edge exists that can be taken if the signal b is received. A third edge exists leading to location *WORKING*. This edge contains the internal signal σ_ϵ that do not need to synchronize with any other TIOA. Further, when taking this edge clock x is reset to zero. Location *WORKING* contains the invariant $x \leq 10$, allowing staying in this location for at most 10 time units. For allowing a better understanding question marks are used for depicting input signal that are received and exclamation marks are used for depicting output signals that are send. Internal signals do not contain exclamation, resp. question marks. Solid lines depict controllable edges and dashed lines depict uncontrollable edges.

$\mathcal{A}_r$ is not progressive because each time $\mathcal{A}_r$ enters location *WORKING*, no signal included in the input I is observable on $\mathcal{A}_r$ for at least two time units. As a result $\mathcal{A}_r$ is not progressive because a reachable state exists, when being in location *WORKING*, where traces of the form $\alpha := 0a0a...$ are not observable. Nevertheless, $\mathcal{A}_r$ is receptive because a strategy A' exists for which holds A' is progressive. Such a strategy exists, e.g., by disabling the

[13] It is not required to compare the strategies. Strategies in [16], like also done in this work are defined to be orthogonal because different strategies of compatible TIOA do not share signals or variables (clocks or attributes) and controllable edges are restricted to those including internal signals.

edge from *READY* to *WORKING*. Because the solid edges include only signals of H and all remaining edges are only enabled if the edges of the original TIOA are enabled, A' is a valid strategy according to Def. 14. This is the case because in each reachable state of A' any trace α, consisting of arbitrary sequences of signals a and b as well as time delays can be observed while excluding deadlocks or zeno traces consisting of internal signals. As a result $\mathcal{A}_r$ obviously is receptive. The previously discussed implications of progressive and receptive behavior can also be found in the example of Fig. 8. As soon as $\mathcal{A}_r$ is able to reach a state where no signal of I can be consumed for longer than zero time units, the property of progressive behavior is violated. For the receptive TIOA $\mathcal{A}_r$ holds that it can be refined by the strategy A', excluding all states where signals in I cannot be observed for more than zero time units, or in other words the example works fine (behaves progressive) when avoiding location *WORKING* (consuming time). The resulting progressive TA, resp. TIOA A' is shown in Fig. 9.

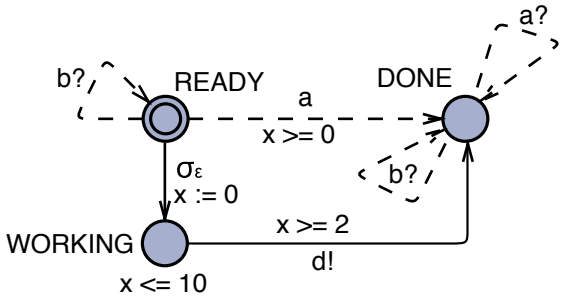


Figure 8: A TIOA $\mathcal{A}_r = (A,I,O,H)$ being receptive and not progressive with $I = \{a,b\}$, $O = \{d\}$, $H = \{\sigma_\epsilon\}$, and a single clock x.

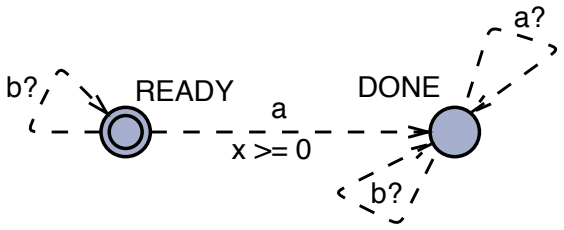


Figure 9: A TA, resp. TIOA A' being progressive with $I = \{a,b\}$, $O = \{d\}$, $H = \emptyset$, and the single clock x.

5 Language Progressive and Language Receptive TIOA

As discussed in Section 4, the properties of progressive and receptive behavior are too demanding and cannot be fulfilled by typical embedded real-time systems using restricted resources (requirement (R2)). This is also reflected by the application example introduced in Sec. 3, where states, respectively locations exist, not allowing receiving an input signal due to the fact that during processing already received signals, no new signals are allowed to arrive. As a consequence, based on the previously defined properties, which exclude deadlocks and zeno-behavior for any possible environment that allows time to diverge, now relaxed conditions are defined. Receptiveness and progressiveness of an automaton $\mathcal{A}$ is expected only w.r.t. the given input language Υ, and not for arbitrary sequences of input signals. The relaxed conditions do not require that a component need to be capable of receiving an arbitrary finite amount of signals in an arbitrary time frame, and as a result support systems with limited resources, fulfilling requirement (R2). Similarly to receptive resp. progressive behavior, it is required that the automaton has no undesired behavior in the form of deadlocks or zeno-behavior for those traces that are included in Υ. Following only TIOA $\mathcal{A}_\Upsilon$ with $traces(\mathcal{A}_\Upsilon) \subseteq \Upsilon$ that do not include a deadlock state or zeno trace are considered to be a valid representation of input languages. In the reminder of this work a zeno and deadlock free TIOA is called a *valid* TIOA. In other words, $\mathcal{A}$ is not able to produce any internal zeno-behavior nor runs into a deadlock, if the environment sends an arbitrary sequence (trace) of signals to $\mathcal{A}$ that is included in Υ.

Definition 15 (Language Progressive)
A TIOA $\mathcal{A} = (A, I, O, H)$ is called Υ-progressive *(written as $\Upsilon\, pro\, \mathcal{A}$) for language Υ with $signals(\Upsilon) \subseteq I$ and Υ not including a deadlock or zeno trace if for all compatible and valid TIOA $\mathcal{A}_\Upsilon$ with $traces(\mathcal{A}_\Upsilon) \subseteq \Upsilon$ holds: $\mathcal{A}_\Upsilon \parallel \mathcal{A}$ does not include a deadlock or zeno trace.*

Informally, a TIOA $\mathcal{A}$ is Υ-progressive if it never generates a zeno or deadlock trace if only sequences of signals are sent to $\mathcal{A}$ that are included in Υ. Υ is an input language of the TIOA $\mathcal{A}$ containing no deadlock and zeno traces. For all TIOA $\mathcal{A}_\Upsilon$ that do only contain signals of the input signals of $\mathcal{A}$, $\mathcal{A}_\Upsilon \parallel \mathcal{A}$ does not include a deadlock or zeno trace. By construction, a zeno trace can only exist in the parallel composition of two TIOA, if at least one TIOA already includes a zeno trace. This is the case because due to synchronization in our model of TIOA transitions can only be removed, but not added. Because zeno traces cannot be invented by the composition and only zeno free TIOA are considered, following only the parallel product of zeno free TIOA is considered. The correctness of a parallel product can only be violated if a deadlock is introduced by the composition of valid TIOA. Thus, following only deadlock traces, resp. deadlock states are considered as undesired properties occurring during composition.

It is also possible to specify a condition based on the states and transitions included in the induced timed system $\mathcal{T}_\mathcal{A}$ of $\mathcal{A}$, indicating if the property of language progressive behavior is violated for a TIOA $\mathcal{A}$ and a language Υ. For this purpose first a so-called transition blocking state of the induced time system of TIOA $\mathcal{A}$ is defined.

Definition 16 (Transition Blocking State)
Given a valid TIOA $\mathcal{A} = (A, I, O, H)$, its induced timed system $\mathcal{T} = (\Sigma, \mathcal{S}, \mathcal{S}^0, T)$ and a language Υ with $traces(\mathcal{A})\restriction I = \Upsilon$. A state $s \in \mathcal{S}$ is called a transition blocking state *if $\exists \alpha \in \Upsilon$, $\alpha = \beta \circ t \circ \gamma$ with $\beta \in traces\_to(s, \mathcal{A})$ and $\neg \exists t_2 \in T(s)$ with:*

- *$t_2\restriction I \vdash t$ if $t \in \Sigma$ (if t is a signal),*
- *$t_2\restriction I \vdash t$ if $t \in \mathbb{R}^+$ (if t is a time delay).*

Intuitively Def. 16 states that a state is reachable in the induced timed system of the TIOA $\mathcal{A}$ that can be reached via a prefix of a single trace α of the language Υ, but continuing this same trace is not possible from this state of the induced timed system of $\mathcal{A}$.

Following in Theorem 5.1 it is shown that a TIOA violates the property of language progressive behavior, iff, a transition blocking state according to Def. 16 exists.

Lemma 5.1 (Language Progressive Condition)
For a given valid TIOA $\mathcal{A} = (A, I, O, H)$, its induced timed system $\mathcal{T} = (\Sigma, \mathcal{S}, \mathcal{S}^0, T)$ and a language Υ with $traces(\mathcal{A})\restriction I = \Upsilon$ holds: $\neg \Upsilon\ pro\ \mathcal{A}$, iff, a transition blocking state *s according to Def. 16 exists for the language Υ.*

Proof 5.1
Two directions are shown separately: 1) Assuming $\neg \Upsilon\ pro\ \mathcal{A}$, with $\mathcal{A} = (A, I, O, H)$ and no state s according to Def. 16 is included in the induced timed system $\mathcal{T}$. By definition a compatible, deadlock and zeno free (valid) TIOA $\mathcal{A}_n$ need to exist with $traces(\mathcal{A}_n) \subseteq \Upsilon$ and $\mathcal{A}_n \parallel \mathcal{A}$ including a deadlock trace. This deadlock trace need to lead to a deadlock state s_d of the induced timed system $\mathcal{T}_{\mathcal{A}_n \parallel \mathcal{A}}$. Thus, there also need to exist a deadlock trace $\alpha_d \in traces\_to(s_d, \mathcal{A}_n \parallel \mathcal{A})$ leading to s_d. For $\alpha_d = \beta$ (β like used in Def. 16) holds there need to exists an α with $\alpha = \beta \circ t \circ \gamma$, $\alpha \in traces(\mathcal{A}_n)$ and α not being a deadlock or zeno trace ($\mathcal{A}_n$ is valid). Because in the deadlock state s_d no transition is observable in that state, also no transition t_2 with $\alpha = \beta \circ t \circ \gamma$ and $t_2\restriction I \vdash t$ is included in the induced timed system of $\mathcal{A}$ leading to a contradiction.

2) Assuming $\Upsilon\ pro\ \mathcal{A}$ and a state according to Def. 16 exists. Because s exists in the induced timed system of $\mathcal{A}$ and s is reachable via a trace $\beta \vdash \alpha$ with $\alpha \in \Upsilon$, a TIOA $\mathcal{A}_\alpha$ exists for which holds $\alpha = traces(\mathcal{A}_\alpha)$, allowing observing only the single trace α. For the parallel product $\mathcal{A}_\alpha \parallel \mathcal{A}$ holds either s is reachable via β or a deadlock occurs (α is the only observable trace). If s is reachable via β, s is a deadlock state by construction or s is not reachable because of a deadlock not allowing reaching s. In both cases $\mathcal{A}_\alpha$ is a counter example indicating that $\neg \Upsilon\ pro\ \mathcal{A}$ leading to a contradiction.

Informally spoken Lemma 5.1 states that if and only if a state s is reachable in the induced timed system $\mathcal{T}$ of $\mathcal{A}$, for that holds it can be reached via a prefix of an input trace taken from the input language Υ, but continuing the same trace is not possible from that state, the property of $\Upsilon\ pro\ \mathcal{A}$ is violated. In other words, if $\mathcal{A}$ is able to prevent the continuation of a trace α included in the input language in a state reachable via a prefix of α, the property of language progressive behavior is violated.

Based on the definition of language progressive behavior the property of language receptive behavior is defined as a relaxed condition of receptive behavior like given in [16].

Definition 17 (Language Receptive)
Let $\mathcal{A} = (A, I, O, H)$ be a valid TIOA. $\mathcal{A}$ is called Υ-receptive *(written as $\Upsilon\, rec\, \mathcal{A}$) if a valid strategy $\mathcal{A}'$ for $\mathcal{A}$ exists with: $\Upsilon\, pro\, \mathcal{A}'$.*

Informally, a TIOA is Υ-*receptive* if it can be refined by a strategy to a Υ-*progressive* TIOA.

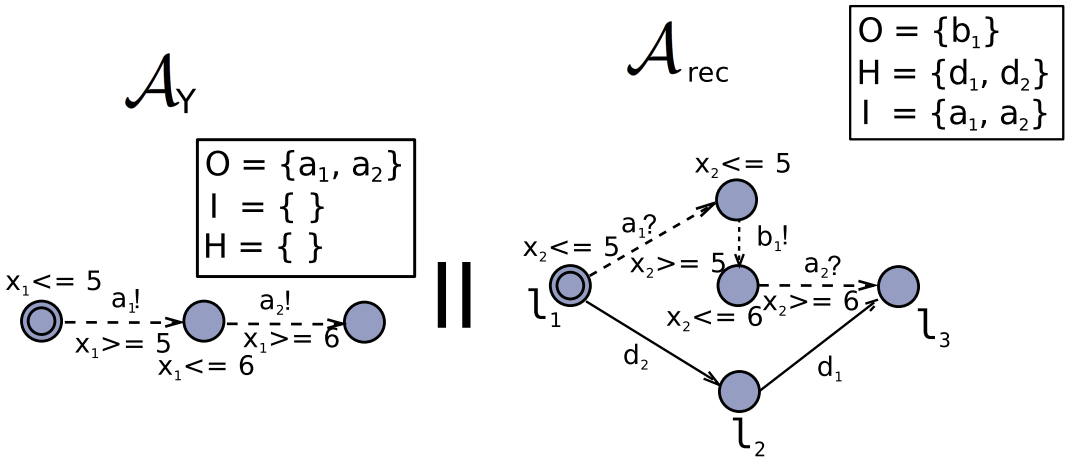


Figure 10: The two TIOA: $A_\Upsilon = (A, I, O, H)$ with $I = \{\}$, $O = \{a_1, a_2\}$, $H = \{\}$ and single clock x_1 and $\mathcal{A}_{rec} = (A, I, O, H)$ with $I = \{a_1, a_2\}$, $O = \{b_1\}$, $H = \{d_1, d_2\}$ and single clock x_2.

For an example of a TIOA $\mathcal{A}_{rec}$ that is $\Upsilon\, rec\, \mathcal{A}_{rec}$ with $\Upsilon = \{\alpha\}$ and $\alpha := 5\ a_1\ 1\ a_2...$ see Fig. 10. $\mathcal{A}_{rec}$ is not language progressive due to the fact that location l_3 is reachable by taking at point in time 0 the edge to location l_2. Trace $\beta := 0\ d_2\ 0\ d_1...$ can be observed leading to l_3. When being in location l_3 a deadlock occurs in case of the parallel product $\mathcal{A}_\Upsilon \parallel \mathcal{A}_{rec}$.

Nevertheless, a strategy $\mathcal{A}'$ for $\mathcal{A}_{rec}$ exists with $\Upsilon\, pro\, \mathcal{A}'$. $\mathcal{A}'$ can be derived by simply disabling the edge between locations l_1 and l_2. As a result, only one trace of the form $\alpha_1 := 5\ \alpha_1\ 0\ b_1\ 1\ \alpha_2...$ remains observable on $\mathcal{A}_{rec}$, leading to location l_3 where time can diverge. For $\mathcal{A}_\Upsilon$ being a compatible TIOA representing Υ holds, no zeno or deadlock trace exist in $\mathcal{A}_\Upsilon \parallel \mathcal{A}'$. In our example this can be obviously seen because Υ only includes a single trace. Thus, the property of language receptive behavior is fulfilled due to the existence of $\mathcal{A}'$.

5.1 Checking Language-Progressive TIOA

Following it is shown how to check if the property $\Upsilon\, pro\, \mathcal{A}$ is fulfilled for a given TIOA $\mathcal{A}^c_\Upsilon$, with $traces(\mathcal{A}^c_\Upsilon) = \Upsilon$. According to Lemma 5.1 this property is fulfilled if no transition block-

ing state according to Def. 16 exists in $\mathcal{A}$, reachable under a trace included in Υ, its input language. It is shown how to transform any given TIOA $\mathcal{A}_\Upsilon$, with $traces(\mathcal{A}_\Upsilon) = \Upsilon$, to a TIOA $\mathcal{A}^c_\Upsilon$, with $traces(\mathcal{A}^c_\Upsilon) = \Upsilon$, fulfilling the property that each transition blocking state of $\mathcal{A}$ leads to at least one deadlock state in the parallel product $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$. Thus, iff, $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ includes a deadlock, the property $\Upsilon \; pro \; \mathcal{A}$ is not fulfilled. This transformation is later on used to create a testbed allowing deciding if the property of language progressive behavior is fulfilled. The overall procedure is sketched in Fig. 11. First, based on a given TIOA $\mathcal{A}_\Upsilon$, representing the input language Υ, a modified version $\mathcal{A}^c_\Upsilon$ is derived representing the same input language. In contrast to the original TIOA $\mathcal{A}_\Upsilon$, $\mathcal{A}^c_\Upsilon$ includes a deadlock if at least one transition blocking state is reachable in $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$. Accordingly, the parallel product $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ is created allowing deciding if $\Upsilon \; pro \; \mathcal{A}$, if (and only if) no deadlock is included in this parallel product.

For being able checking if the property $\Upsilon \; pro \; \mathcal{A}$ is fulfilled for a given TIOA $\mathcal{A}^c_\Upsilon$, with $traces(\mathcal{A}^c_\Upsilon) = \Upsilon$, first a condition under that a violation of language progressive behavior can be detected in form of a deadlock when building the parallel product $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ is defined. In Theorem 5.1 a property is defined, if fulfilled, $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ includes a deadlock, if (and only if) $\Upsilon \; pro \; \mathcal{A}$ is violated. Afterwards it is shown how to transform any given TIOA $\mathcal{A}_\Upsilon$, with $traces(\mathcal{A}_\Upsilon) = \Upsilon$, to a TIOA $\mathcal{A}^c_\Upsilon$ with $traces(\mathcal{A}^c_\Upsilon) = \Upsilon$, fulfilling the property defined in Theorem 5.1. This transformation is later on used to create a testbed, allowing deciding if the property of language progressive behavior is fulfilled. The only precondition is that a TIOA is available being a valid representation of the input language. Such a TIOA can be found in a component based architecture by the neighbored component sending input signals to $\mathcal{A}$.

First a *transition enforcing state* is defined. A *transition enforcing state* is a state where either exactly one single signal can be send or time is allowed to pass.

Definition 18 (Transition Enforcing State)
Let $\mathcal{A}^c_\Upsilon$ be a valid TIOA and $\mathcal{T}_{\Upsilon^c} = (\Sigma_{\Upsilon^c}, \mathcal{S}_{\Upsilon^c}, \mathcal{S}^0_{\Upsilon^c}, T_{\Upsilon^c})$ be the induced timed system of $\mathcal{A}^c_\Upsilon$. $s \in \mathcal{S}_{\Upsilon^c}$ is called a transition enforcing state *if $\forall t \in T_{\mathcal{A}_{\Upsilon^c}}(s)$ holds: either $t = s \xrightarrow{\sigma_\epsilon} s'$ containing the empty signal or $t = s \xrightarrow{\sigma} s'$ with $\sigma \in \Sigma_{\Upsilon^c}$ and $|T_{\mathcal{A}_{\Upsilon^c}}(s)| = 1$ or $t = s \xrightarrow{\tau} s'$ with $\tau \in \mathbb{R}^+$.*

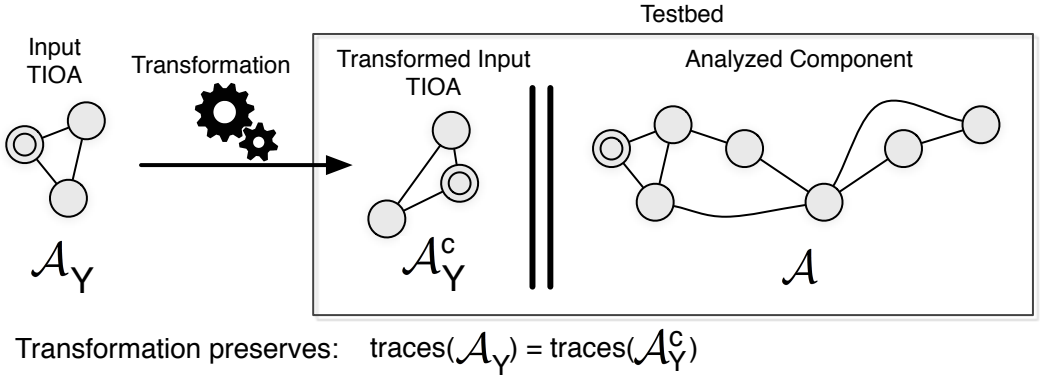


Figure 11: The schematic description of the analysis for language progressive behavior.

Based on the definition of a *transition enforcing state* a *transition enforcing TIOA* is defined. Such a *transition enforcing TIOA* is defined based on all observable states and transitions (discrete or continuous) of its induced timed system.

Definition 19 (Transition Enforcing TIOA)
Let $\mathcal{A}^c_\Upsilon$ be a valid TIOA and $\mathcal{T}_{\Upsilon^c} = (\Sigma_{\Upsilon^c}, \mathcal{S}_{\Upsilon^c}, \mathcal{S}^0_{\Upsilon^c}, T_{\Upsilon^c})$ be the induced timed system of $\mathcal{A}^c_\Upsilon$. $\mathcal{A}^c_\Upsilon$ is called a transition enforcing TIOA *if $\forall \beta, s, t$ with $\beta \in traces\_to(s, \mathcal{A}^c_\Upsilon)$, $s \in \mathcal{S}^c_\Upsilon$ and $t \in T^c_\Upsilon(s)$ holds: $\exists s_2 \in \mathcal{S}^c_\Upsilon$ with $\beta \in traces\_to(s_2, \mathcal{A}^c_\Upsilon)$ being a transition enforcing state according to Def. 18 with $t \in T^c_\Upsilon(s_2)$.*

More informally spoken Def. 19 can be described as follows. Assuming a TIOA $\mathcal{A}^c_\Upsilon$ representing a language. If for this automaton holds: for any state s, being reachable via a trace β allowing observing transition t (beneath other transitions in this state), Def. 19 holds, iff, a state s_2 is reachable via β only allowing observing transitions which are equal to t or which are a prefix of t.

It needs to be considered that a transition enforcing TIOA is not necessarily a deterministic TIOA. Beneath the transition enforcing states additional states with nondeterministic transition relations are allowed to exist. Further, the successor states can potentially be nondeterministically chosen, also in case of a transition enforcing state.

Based on Def. 19 a condition can be defined allowing deciding if the property of language progressive behavior is fulfilled.

Theorem 5.1 (Language Progressive Test)
Let $\mathcal{A} = (A, I, O, H)$ be a valid TIOA and $\mathcal{A}^c_\Upsilon = (A^c_\Upsilon, I^c_\Upsilon, O^c_\Upsilon, H^c_\Upsilon)$ be a valid transition enforcing TIOA according to Def. 19 with $I = O^c_\Upsilon$, $I^c_\Upsilon = \emptyset$ and $\Upsilon = traces(\mathcal{A}^c_\Upsilon)$.

It holds $\neg \Upsilon \; pro \, \mathcal{A}$, iff, $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ includes a deadlock.

Proof 5.2
(sketch) Let $\mathcal{T}_\mathcal{A} = (\Sigma_\mathcal{A}, \mathcal{S}_\mathcal{A}, \mathcal{S}^0_\mathcal{A}, T_\mathcal{A})$ be the induced timed system of $\mathcal{A}$ and $\mathcal{T}_{\Upsilon^c} = (\Sigma_{\Upsilon^c}, \mathcal{S}_{\Upsilon^c}, \mathcal{S}^0_{\Upsilon^c}, T_{\Upsilon^c})$ be the induced timed system of $\mathcal{A}^c_\Upsilon$. Two different cases can exist if Theorem 5.1 is not fulfilled: 1) $\neg \Upsilon \; pro \, \mathcal{A}$ and no deadlock is included in $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ and 2) $\Upsilon \; pro \, \mathcal{A}$ and a deadlock occurs in $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$.

Assuming 1): $\neg \Upsilon \; pro \, \mathcal{A}$ and no deadlock is included in $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$. Because of Lemma 5.1 holds: $\exists s_\mathcal{A} \in S_\mathcal{A}$, $\alpha \in \Upsilon$ with $\alpha = \beta \circ t \circ \gamma$ and $\neg \exists t_\mathcal{A} \in T_\mathcal{A}(s_\mathcal{A})$ with $t_\mathcal{A} \vdash t$. Because $\alpha \in \Upsilon$ and $traces(\mathcal{A}^c_\Upsilon)$, in $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ state $s_\mathcal{A}$ need to be reachable for $\mathcal{A}$ in this parallel product. There need to exists a state of $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ where $\mathcal{A}$ is in state $s_\mathcal{A}$ and $\mathcal{A}^c_\Upsilon$ is in state s^c_Υ with $t^c_\Upsilon \in T^c_\Upsilon(s^c_\Upsilon)$ and $t^c_\Upsilon \vdash t$ (because $\alpha \in \Upsilon = traces(\mathcal{A}^c_\Upsilon)$). Because $\mathcal{A}^c_\Upsilon$ is a transition enforcing TIOA, a transition enforcing state $s2^c_\Upsilon \in \mathcal{S}^c_\Upsilon$ of TIOA $\mathcal{A}^c_\Upsilon$ is reachable in the parallel product under the same circumstances, only containing outgoing transitions of the form $t^c_\Upsilon = s \xrightarrow{a} s'$. According to Def. 18 such a transition of a transition enforcing state can contain either a signal or a delay (either $a \in \Sigma_\mathcal{A}$ or $a \in \mathbb{R}^+$). In case of a signal a single outgoing transition exists for this state with $t^c_\Upsilon = t$, immediately leading to a deadlock (t cannot be consumed by $\mathcal{A}$ in this state of the parallel product.). In case of a continuous transition, all outgoing transitions have only delays > 0 and as a result, again, t^c_Υ is of the form $^c_\Upsilon \vdash t$, leading to a

deadlock in the parallel product. As a consequence a deadlock need to occur and leading to a contradiction.

Assuming 2): $\Upsilon \; pro \, \mathcal{A}$ *and a deadlock occurs in* $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$*. Because* $\mathcal{A}^c_\Upsilon$ *is a witness violating Def. 15,* $\Upsilon \; pro \, \mathcal{A}$ *cannot be fulfilled leading to a contradiction.*

In other words, if the environment of $\mathcal{A}^c_\Upsilon$ (in this case the environment of $\mathcal{A}^c_\Upsilon$ is $\mathcal{A}$) blocks any transition of $\mathcal{A}^c_\Upsilon$ at any time, a deadlock is reachable in the induced timed system. Because the involved TIOA are deadlock free in isolation, this deadlock allows deciding if the property of language progressive behavior is fulfilled. Subsequently it is shown how to transform any given TIOA representing an input language to a TIOA fulfilling the required properties of Def. 19, without removing or adding any trace. In this case, it is possible checking for language progressive behavior according to Theorem 5.1.

5.2 Creating a Testbed

It is following described how to create a testbed allowing deciding if a transition blocking state according to Def. 16 is included in the induced timed system of $\mathcal{A}$. It is shown how to construct a modified version $\mathcal{A}^c_\Upsilon$ of $\mathcal{A}_\Upsilon$ for that holds 1) $traces(\mathcal{A}_\Upsilon) = traces(\mathcal{A}^c_\Upsilon)$ and 2) for each transition blocking state s of the induced timed system $\mathcal{T}_\mathcal{A}$ of $\mathcal{A}$, being reachable via an input trace $\alpha \in \Upsilon$, a deadlock is included in the parallel product $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$.

According to Def. 16 it is distinguished between the two different types of transitions in form of discrete and continuous transitions. First discrete transitions $s \xrightarrow{\sigma} s'$ of $\mathcal{T}_{\mathcal{A}_\Upsilon}$ are considered, where $\sigma \in \Sigma$ is a signal. Such a transition can exist in a state of the induced timed system $\mathcal{T}_{\mathcal{A}_\Upsilon}$ while being blocked in the parallel product $\mathcal{A}_\Upsilon \parallel \mathcal{A}$, indicating the existence of a transition blocking state of the TIOA $\mathcal{A}$. As an example see the parallel product of two valid and compatible TIOA shown in Fig. 12. While the TIOA $\mathcal{A}_\Upsilon = (A_\Upsilon, I_\Upsilon, O_\Upsilon, H_\Upsilon)$ with $O_\Upsilon = \{a\}, I_\Upsilon = H_\Upsilon = \emptyset$ on the left has the possibility to send signal a at point in time two, the valid and compatible TIOA $\mathcal{A}$ on the right of Fig. 12 is blocking this discrete transition till point in time three. Thus, a transition blocking state needs to exist in $\mathcal{A}$.

Subsequently the case of a discrete transition $t = s \xrightarrow{\sigma} s'$ of the induced timed system $\mathcal{T}_{\mathcal{A}_\Upsilon}$ is considered with $s = \langle l, v \rangle$ and $s' = \langle l', v' \rangle$. l is the location of state s and v is the assignment of clock variables in this state. Accordingly l' is the location of s' and v' is the assignment of clock variables. Following it is shown how to create for each discrete transition t of each state $s = \langle l, v \rangle$ of the induced timed-system of $\mathcal{A}_\Upsilon$, a second state $s_2 = \langle l_{urgent}, v \rangle$, being reachable via the same trace, allowing only taking discrete transition $t_2 = s_2 \xrightarrow{\sigma} s'$, including the same signal σ. Each edge of the TIOA $\mathcal{A}_\Upsilon$ including a signal $a \in O_\Upsilon$ is replaced by two edges and an additional *urgent* location in between. In a timed automaton, an urgent location is a location where time is not allowed to diverge. Thus, extended TA are used like defined in Def. 2. The replacing works as follows. Each found edge $e = (l, a, \varphi, \lambda, l')$ including a signal $a \in O_\Upsilon$, is replaced by $e = (l, \epsilon, \varphi, \lambda, l_{urgent})$, leading to the newly created urgent location. Additionally an edge $e_a = (l_{urgent}, a, \emptyset, \emptyset, l')$ is created including an empty guard, the signal a and an empty set of updated clocks leading to l'. In Fig. 13 the resulting changed

TIOA $\mathcal{A}^c_\Upsilon$ is shown. The original TIOA $\mathcal{A}_\Upsilon$ is shown on the left of Fig. 12. By construction no trace is added by this modification, due to fact that taking any newly created transition to the urgent location is possible in the same absolute point of time τ when the original TIOA $\mathcal{A}_\Upsilon$ is able to send the associated signal (the transition to the urgent location is reachable via an ϵ, the empty signal not occurring in any trace). Thus, no trace is added or removed by this modification and it holds: $traces(\mathcal{A}_\Upsilon) = traces(\mathcal{A}^c_\Upsilon)$. Because the edge is leading to an urgent location, time is not allowed to pass and the urgent location need to be left via the only outgoing transition sending the signal at point in time τ. Thus, for each state of the induced timed system where a discrete transition can take place, a state exists (a transition enforcing state) where only this transition can be taken. As a consequence, by construction each time a transition blocking state is reached in the parallel product $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ in form of a discrete transition, a deadlock need to occur. With $mod_\sigma(\mathcal{A}_\Upsilon)$ the operation realizing the previously described modifications on the TIOA $\mathcal{A}_\Upsilon$ is denoted.

Fig. 12 also shows an example where $\mathcal{A}$ is blocking a continuous transition of the induced timed system of $\mathcal{A}_\Upsilon$. This is the case because $\mathcal{A}_\Upsilon$ can stay for 10 time units in the initial location, but in the parallel product $\mathcal{A}$ is blocking the progress of time when clock d is equal to 5. As a result $\mathcal{A}$ enforces $\mathcal{A}_\Upsilon$ taking a discrete transition, e.g., not allowing taking the continuous transition $s \xrightarrow{\tau} s'$ with $\tau = 6$. $s = \langle l, v\rangle$ consists of location l as well as the assignment of clock variables v, and $s' = \langle l, v'\rangle$ consists of location l and clock variable assignment v'. According to the definition of an induced timed system like given in Sec. 2, continuous transitions do not change the location parameter l. Thus, states s and s' have the same location parameter.

According to Def. 16 such blocking of continuous transitions is caused by a transition blocking state and violates the property of language progressive behavior. This is the case because a trace $\alpha = \beta \circ t \circ \gamma \in \Upsilon$ exists, with $\alpha = 6a...$, $\beta = 5$ and t being a continuous transition including a $\tau > 0$ for that holds: in state $s_\mathcal{A}$ of $\mathcal{A}$ (at the initial location) at point in time 5, no transition t_2 exists in that state of $\mathcal{A}$ with $t_2 \vdash t$. Following it is shown how to modify $\mathcal{A}_\Upsilon$ such that if a continuous transition $s \xrightarrow{\tau} s'$ of $\mathcal{T}_{\mathcal{A}_\Upsilon}$ is enabled in a state reachable via a trace β, a second state is reachable via β only allowing taking continuous transitions. In this case $\mathcal{A}_\Upsilon$ is in a state respectively a location l_i where the invariants $\mathcal{I}(l_i) = \{x_{i,1} \sim c_{i,1}, ..., x_{i,n} \sim c_{i,n}\}$ with $\sim \in \{\leq, \geq, \}$ allow the progress of time for $\tau > 0$. Thus, it is possible to leave this location via an edge for that holds: the guard $\{x_{i,1} \sim c_{i,1}, ..., x_{i,n} \sim c_{i,n}\}$ is fulfilled, or in other word no upper bound of the invariant associated with the location of this state is passed. $\mathcal{A}_\Upsilon$ is modified in such a way that for each location l_i of $\mathcal{A}_\Upsilon$ a copy $l_{(i,c)}$ is added. Now for each pair $l_i, l_{i,c}$ an edge $e_{i,c} = (l_i, \epsilon, \varphi_{i,c}, \lambda, l_{i,c})$ is added, with the guard $\varphi_{i,c} = \{x_{i,1} \sim c_{i,1}, ..., x_{i,n} \sim c_{i,n}\}$ being equal to the invariant of the original location. Thus, this newly created location can be reached in the induced timed system of $\mathcal{A}_\Upsilon$ as well as in $\mathcal{A}_\Upsilon \parallel \mathcal{A}$, as long as $\mathcal{A}_\Upsilon$ is able to stay in any state $s = \langle l_i, v\rangle$, allowing taking continuous transitions only (allow time to pass). Further a second edge $e_{c,i} = (l_{i,c}, \epsilon, \varphi_{c,i}, -, l_i)$ is added, with $\varphi_{c,i} = \{x_{i,1} \geq c_{i,1} \vee ... \vee x_{i,n} \geq c_{i,n}\}$,[14] leading back to the original location. See the previously described example of Fig. 12 where the continuous transition $s \xrightarrow{\tau} s'$, with $s = \langle l_0, \{c = 5\}\rangle$, $s' = \langle l_0, \{c = 6\}\rangle$ and $\tau = 1$ being prevented in the induced timed system $\mathcal{T}_{\mathcal{A}_\Upsilon \parallel \mathcal{A}}$. A copy $l_{0,c}$

[14] By choosing the guard of the outgoing transition using the same values for the bounds, but with $\geq$ instead of $\leq$ (compared to the original invariant), the transition can be taken if for at least one equation of the invariant and the guard holds: $c \leq n \wedge c \geq n$ being equivalent to $c = n$, with n being an upper bound of the invariant.

of l_0 is added to $\mathcal{A}_\Upsilon$, using the invariants of l_0. Further edge $e_{0,c} = (l_0, \epsilon, \{c \leq 10\}, \emptyset, l_{0,c})$ and edge $e_{c,0} = (l_{0,c}, \epsilon, \{c \geq 10\}, \emptyset, l_0)$ is added. The result is shown in Fig. 14. Unfortunately this modification introduces zeno behavior as follows. Consider the example shown in Fig. 14. The newly created edges from and to the copied location $l_{0,c}$ can be taken infinitely often at the point in time when clock c has the value 10. This is the case because the added edges do not require time to pass in case clock c has the value 10. For avoiding this effect an additional attribute variable $count$ is added to the TIOA. This variable is used inside guards and updates accordingly to prevent zeno behavior like shown in Fig. 14. As result, the newly created edges can only be taken when re-entering the original location (e.g., location l_0). In the remainder of this work these additional attribute variable are skipped from the used figures to allow a better understanding. With $mod_\tau(\mathcal{A}_\Upsilon)$ the operation realizing the previously described modifications related to continuous transition is denoted, applied on the TIOA $\mathcal{A}_\Upsilon$.

To conclude, $\mathcal{A}_\Upsilon$ is modified such that it can decide if it will only take continuous transitions (allowing time to diverge) till the upper bound of the invariant, restricting the possible continuous transitions of a location, is reached. The made modifications do not add or remove any traces to $traces(\mathcal{A}^c_\Upsilon)$ which have not been previously included in $traces(\mathcal{A}_\Upsilon)$. This is the case because guards and invariants of the added locations and edges are chosen, such that $\mathcal{A}^c_\Upsilon$ can decide to prevent sending any signal for a time interval, iff, $\mathcal{A}_\Upsilon$ is able taking only continuous transitions for the same time interval (not sending any signal). Further holds, if $\mathcal{A}_\Upsilon$ do not need to send a signal for a time period greater than zero, $\mathcal{A}^c_\Upsilon$ is also able to enter a state where synchronization is not able to force to send a signal for at least the same time period. Thus, when being in a newly created location in $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ and $\mathcal{A}$ is blocking a discrete transition, a deadlock needs to occur.

The resulting testbed TB is defined as the parallel composition of $\mathcal{A}^c_\Upsilon$ and $\mathcal{A}$. The construction of the testbed is summarized in the following definition.

Definition 20 (Testbed)
Let $\mathcal{A}_\Upsilon = (A_\Upsilon, I_\Upsilon, O_\Upsilon, H_\Upsilon)$ and $\mathcal{A} = (A, I, O, H)$ be two compatible and valid TIOA with $O_\Upsilon = I$. The testbed for *$\mathcal{A}_\Upsilon$ and $\mathcal{A}$ is defined by the TIOA*

$$TB(\mathcal{A}_\Upsilon, \mathcal{A}) = \mathcal{A}^c_\Upsilon \parallel \mathcal{A}$$

with:

- $\mathcal{A}^c_\Upsilon = (A^c_\Upsilon, I_\Upsilon, O_\Upsilon, H_\Upsilon) = mod_\tau(mod_\sigma(\mathcal{A}_\Upsilon))$,
- $I = O_\Upsilon$, $H = \emptyset$, $O = \emptyset$ *and*
- $traces(\mathcal{A}_\Upsilon) = traces(\mathcal{A}^c_\Upsilon)$
- $\forall t \in T_\mathcal{A}(s)$ *with* $\alpha \in traces\_to(s, \mathcal{A})$*:* $\exists s'$ *with* $\alpha \in traces\_to(s', \mathcal{A}^c_\Upsilon)$ *being a transition enforcing state for transition* t.

Corollary 5.1
Given two valid and compatible TIOA $\mathcal{A}_\Upsilon$ and $\mathcal{A}$, each testbed according to Def. 20 allows deciding if the property $\Upsilon \; pro \; \mathcal{A}$ is fulfilled.

Proof 5.3
(sketch) By construction the TIOA $\mathcal{A}^c_\Upsilon \parallel \mathcal{A}$ includes a deadlock, iff, $\mathcal{A}$ has a transition blocking state according to Def. 16 ($\mathcal{A}$ and $\mathcal{A}^c_\Upsilon$ are valid). Because the testbed only contains a deadlock iff a transition blocking state is included, the testbed allows to decide if the property of language progressive behavior is fulfilled.

According to Corollary 5.1 the testbed allows us deciding whether $\Upsilon \; pro \, \mathcal{A}$ by checking freedom of deadlock behavior of the testbed. Thus, for an input language Υ, represented by a TIOA $\mathcal{A}^c_\Upsilon$ with $traces(\mathcal{A}^c_\Upsilon) = \Upsilon$, we are able to automatically decide whether the language progressiveness is fulfilled. For the example shown in Fig. 12 this property is not fulfilled. This is the case because discrete as well as continuous transitions are blocked like previously discussed. For the TIOA shown in Fig. 16 the property $\mathcal{A}_\Upsilon \; pro \, \mathcal{A}$ is fulfilled for the TIOA $\mathcal{A}_\Upsilon$ shown on the left of Fig. 12.

In contrast to progressive or input enabled behavior, like used in [16] or [10], language progressive TIOA are not required being able to consume any amount of signals in any time. For example the TIOA shown in Fig. 16 is allowed to stay in the initial location for one time unit not being able to consume any signal. Thus, also TIOA are supported that are not able to consume signals for a given time interval, allowing representing the behavior of components like described in Sec. 3.

Now the example of the Engine-Control model introduced in Sec. 3 is considered. Lets consider the two components *EngineModel* and *FuelSensor*. Component *EngineModel* is sending signals to component *FuelSensor* (signals of type *raw*) and accordingly it is of interest weather the TIOA $\mathcal{A}_{FS}$ is language progressive for the language defined by the TIOA *EngineModel*. By applying the previously described steps on the TIOA representing the component *EngineModel*, a TIOA is achieved fulfilling the conditions like defined in Theorem 5.1. The resulting TIOA $\mathcal{A}^c_{EM}$ is shown in Fig. 27. The input signal *fuel_rate* of component *EngineModel* has been removed before applying the modifications. This is the case because the input signal of component *EngineModel* is not synchronized in this test and not part of the language Υ (containing input signals of component *FuelSensor*, resp. output signals of component *EngineModel*).

The TIOA $\mathcal{A}^c_{EM}$ is created as follows. First, all input-signals are removed (signal *fuel_rate*). Second, for each location, a copied location is created that allows only to be left if the invariant of the original location does expire. In Fig. 27 the added locations (*Init_clone*, *wait_clone*, *CRV_clone* and the location without a name) do have the name of the original ones, followed

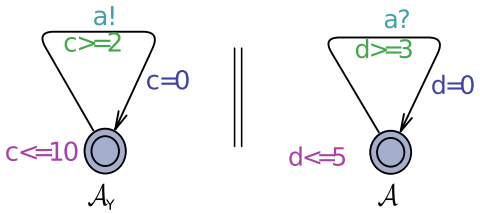


Figure 12: Blocking a discrete transition included in $\mathcal{T}_{\mathcal{A}_\Upsilon}$

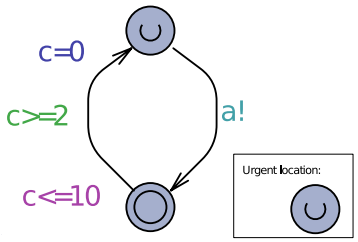


Figure 13: Modified TIOA leading to a deadlock in the case a discrete transition is blocked.

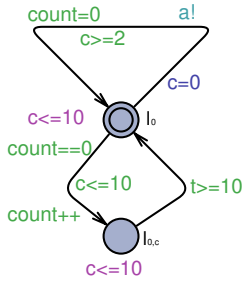


Figure 14: Modified TIOA leading to deadlock in the case a continuous transition is blocked.

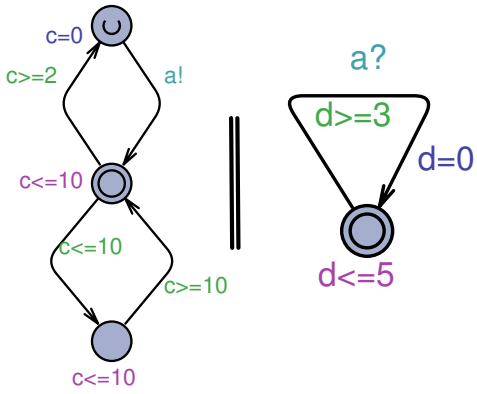


Figure 15: Example of a testbed allowing detecting that the TIOA on the right is not language progressive for the langue defined by the TIOA on the left.

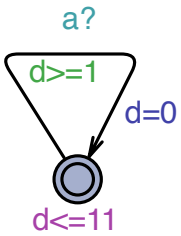


Figure 16: $\mathcal{A}$ being language progressive for the language defined by the TIOA $\mathcal{A}_\Upsilon$ shown in the left of Fig. 12.

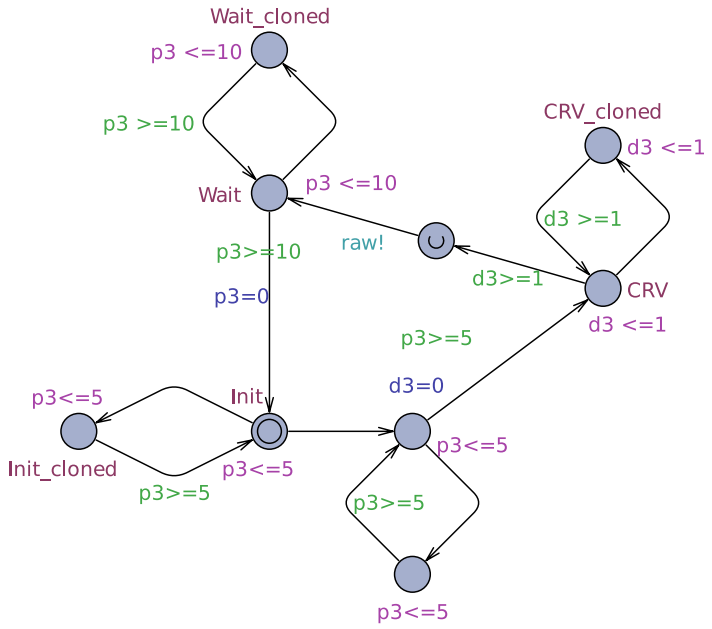
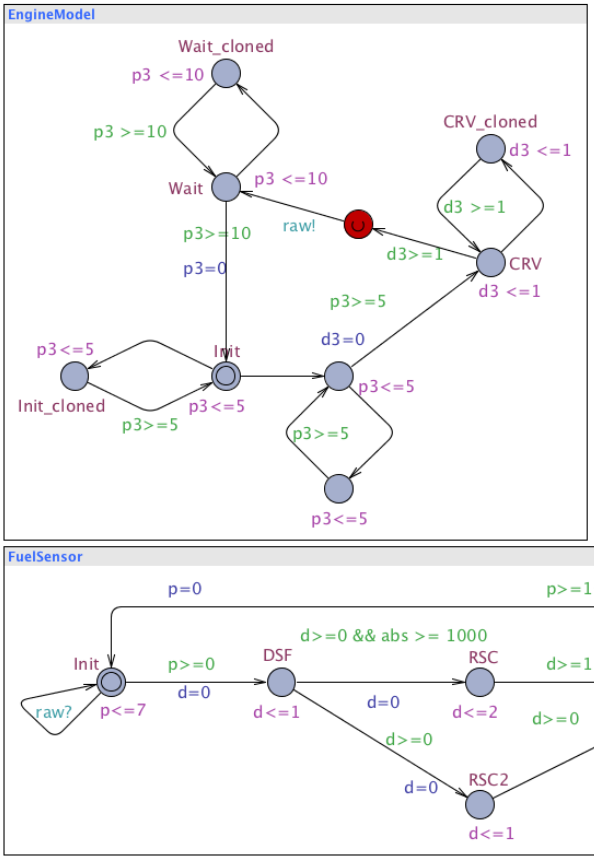


Figure 17: TIOA $\mathcal{A}^c_{EM}$ for testing language progressiveness of TIOA $\mathcal{A}_{FS}$ representing component *FuelSensor* like shown in Fig. 5.

by the postfix *_clone*. The only outgoing transition of each added location, back to the original location, has a modified guard like previously described. For the only edge including an output signal, an urgent location has been created according to the previously described procedure.

Building the parallel product $\mathcal{A}^c_{EM} \parallel \mathcal{A}_{FS}$ allows to search for deadlocks, indicating that the property of language progressive behavior is violated. For this purpose the tool UPPAAL [8] is used. UPPAAL supports model checking timed automata. Due to the fact that each TIOA is also a TA, UPPAAL can be used for this purpose. The semantics of a deadlock like discussed in this work and a deadlock like defined in UPPAAL is slightly different. For more information about these differences as well as how to use UPPAAL for searching deadlocks like defined in this work, compare Appendix E. Further, UPPAAL supports the generation of examples, resp. counterexamples for properties that are fulfilled respectively violated (depending on the property defined). The model checking capabilities of UPPAAL are applied to find out if the individual components include a deadlock. Only TA are considered that are deadlock and zeno free (valid TIOA) to be reasonable. UPPAAL is used to verify that the components of the application example described in Sec. 3 are deadlock and zeno free (well-formed). When applying model-checking using UPPAAL the resulting state-space of the isolated individual components turned out to be rather small, as expected.

Afterwards model checking has been applied on the parallel product $\mathcal{A}^c_{EM} \parallel \mathcal{A}_{FS}$, searching for a deadlock violating the property of language progressive behavior. A counterexample has been generated by UPPAAL (see Fig. 18), where $\mathcal{A}^c_{EM}$ is in a state where signal *raw* can be send by $\mathcal{A}^c_{EM}$, but $\mathcal{A}_{FS}$ is not able to receive this signal in the parallel product.



FuelSensor.p = 6
FuelSensor.d ∈ [0,1]
FuelSensor.abs = 6
FuelSensor.p - FuelSensor.d ∈ [5,6]
FuelSensor.p = FuelSensor.abs
FuelSensor.abs - FuelSensor.d ∈ [5,6]

Figure 18: Counter example in UPPAAL, violating the property of language progressive behavior: $\mathcal{A}_{FS}$ reaching a state where the signal *raw* cannot be consumed while $\mathcal{A}_{EM}$, resp. $\mathcal{A}^{c}_{EM}$ is able to send it.

It turned out that a path exists leading to a state s, of the induced timed system of $\mathcal{A}_{FS}$, where $\mathcal{A}_{EM}$, resp. $\mathcal{A}^c_{EM}$ is able to send the signal *raw* but $\mathcal{A}_{FS}$ is not able to receive it. Thus, the property of language progressive behavior is violated according to Lemma 5.1 and Theorem 5.1. The derived counterexample consists of the current locations of the involved TIOA as well as the possible clock valuations in this state. UPPAAL provides a counterexample in form of a so-called symbolic state, where all possible clock assignments are represented in form of equations over clock variables (for more information about the symbolic representation compare [14, 18]). In our example UPPAAL provides a symbolic state representing an example for a deadlock like shown in Fig. 18, consisting of location *wait* of TIOA $\mathcal{A}_{FS}$ and constraints about the clocks p, abs and d[15] is of interest:

- $p = 6$,
- $d \in [0, 1]$,
- $abs = 6$,
- $p - d \in [5, 6]$,
- $p = abs$ and
- $abs - d \in [5, 6]$.

Accordingly, all concrete clock assignments fulfilling these equations are deadlock states according to Def. 12. As a consequence a state $s_1 = \langle l, v \rangle$ exists, with $l =$ *RSC* and v, the clock valuation, such that $p = 6 \wedge d \in [0, 1] \wedge abs = 6 \wedge p - d \in [5, 6] \wedge p = abs \wedge abs - d \in [5, 6]$, reachable under the parallel product of $\mathcal{A}^c_{EM} \parallel \mathcal{A}_{FS}$, violating the property of language progressive behavior ($traces(\mathcal{A}^c_{EM})\, pro\, \mathcal{A}_{FS}$).

Detecting violations of language progressive behavior as well as identifying states is the first step for achieving a deadlock free and valid architecture. Removing such violations is the next step. In the following section it is shown how to use timed games for deriving configurations that avoid such states. Fortunately, techniques and tools already exist (see [7, 9]), allowing searching for strategies avoiding states.

5.3 Checking Language Receptive TIOA

Checking if a TIOA $\mathcal{A} = (A, I, O, H)$ is language progressive $\Upsilon\, pro\, \mathcal{A}$ according to Def. 15 for a language Υ, represented by an automaton $\mathcal{A}^c_\Upsilon$, can be achieved using the testbed described in the previous section by searching for deadlock states.

Assuming the case that the resulting TB includes a deadlock ($\neg(\Upsilon\, pro\, \mathcal{A})$). According to Def 17 a strategy $\mathcal{A}'$ can exists with $\Upsilon\, pro\, \mathcal{A}'$, if $\mathcal{A} = (A, I, H, O)$ is language receptive for Υ. Following it is described how to achieve strategies $\mathcal{A}'$ for $\mathcal{A}$, fulfilling the property $\Upsilon\, pro\, \mathcal{A}'$.

[15] Constraints about clocks of the TIOA representing the *EngineModel* ($p3$ and $d3$) are not considered because in the used framework only the state-information of $\mathcal{A}_{FS}$.

$\mathcal{A}$ is used for defining a timed game where at most those edges containing signals of H (or no signal) are controllable like described in Sec. 2.2. All modified TA that are a valid strategy according to Def. 14 can be used to remove undesired behavior, e.g., in case of deadlocks.

The previously described example of the *FuelSensor* is investigated, where the testbed shown in Fig 18 indicates that the property of language progressive behavior is violated. The derived counterexamples, in case of discovered deadlock-state s_1 of the testbed, respectively the state information of the deadlock-state that can be observed on $\mathcal{A}_{FS}$ only, can be used to search for a strategy $\mathcal{A}'_{FS}$ of the TIOA $\mathcal{A}_{FS}$, avoiding this state. For this purpose a goal need to be defined for a timed two player game. Such a goal is defined by the state $s = \langle l, v \rangle$ with $l =$ *wait* and v, being a variable valuation, such that $p = \wedge d \in [0,1] \wedge abs = 6 \wedge p - d \in [5,6] \wedge p = abs \wedge abs - d \in [5,6]$ (p, abs and d are clock variables). The goal is defined as follows: avoid all states s_1 observable on $\mathcal{A}_{FS}$, for that hold: the current location is equal to l and for the possible clock assignments holds $p = \wedge d \in [0,1] \wedge abs = 6 \wedge p - d \in [5,6] \wedge p = abs \wedge abs - d \in [5,6]$.

One valid strategy avoiding this state is the TIOA $\mathcal{A}'_{FS}$ shown in Fig. 19. The only difference compared to the original TIOA $\mathcal{A}_{FS}$, like shown in Fig. 5, is, that the controllable transition from the initial location to location *DSF* can only be taken if the clock variable p has a value greater or equal to 7. Thus, a state according to s can no longer be reached, due to the fact that p is always greater than 7 when entering location *wait*.

While the previously shown strategy ($\mathcal{A}'_{FS}$) is derived manually, tool support exists for automation support. Timed two player games can be formulated within the tool UPPAAL TIGA (see [7]). UPPAAL TIGA is a branch of UPPAAL, allowing deciding if a strategy exists as well as allowing to automatically synthesizing a strategy. For this purpose edges included in the TIOA $\mathcal{A}_{FS} = (A_{FS}, I_{FS}, H_{FS}, O_{FS})$ are divided into controllable and non-controllable. Like described in Sec. 4, only edges not including signals of I_{FS} and O_{FS}, not containing input or output signals, are allowed to be controllable in our framework of TIOA. The resulting TIOA is shown in Fig. 20, where all dashed edges are defined to be uncontrollable and all solid edges are defined to be controllable. In this example only the edge between location *Init* and location *DSF* is defined to be controllable. This edge is associated with the decision at which relative point in time within each period of 10 ms the functionality, represented by the locations *DSF*, *RSC* and *RSC2*, is triggered. At a more technical level this decision can be seen as a scheduling configuration, deciding what is the last possible time within each period

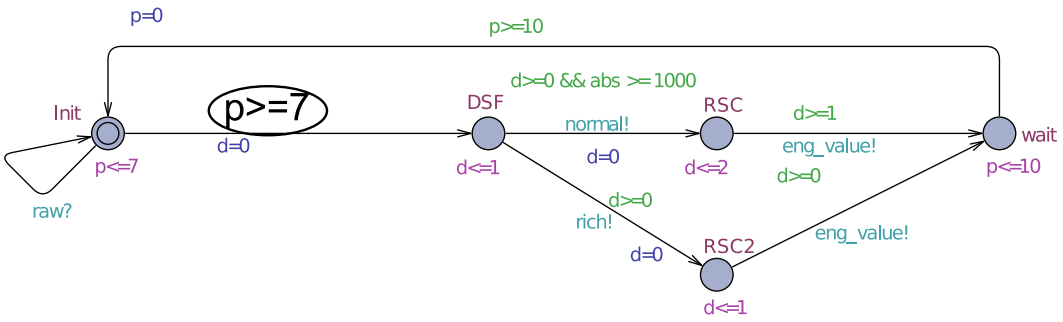


Figure 19: The TIOA $\mathcal{A}'_{FS}$ representing a valid strategy for $\mathcal{A}_{FS}$.

of 10 ms to trigger the functionality included in component *FuelSensor*. In the original TIOA depicted in Fig. 5, this relative point in time is reached in case the clock variable p, included in the invariant of location *Init*, has reached the value 7. Within the timed game this edge is defined to be controllable, allowing searching for a strategy where this relative point in time can be chosen arbitrary as long as not violating the invariant of location *Init*. Further, the goal of the timed game is defined as follows: all states $s = \langle l, v \rangle$ for which hold s is in location l=*DSF* with a variable assignment of v like previously described, have to be avoided. In other words, the goal of the game is not to reach a state according to s_1.

UPPAAL TIGA is trying to find a strategy fulfilling this goal and as a result decides if such a strategy can exist. Further, UPPAAL TIGA allows to synthesizing a strategy that fulfills this goal. Because in UPPAAL TIGA strategies are not defined directly as a timed automaton, the semantics are slightly different such that strategies are functions that, based on the full history, specify the next allowed transition. The strategies used in this work are memoryless and defined based on TA. As a consequence the synthesized strategies cannot be directly used in any case as a valid strategy like defined in Def. 14. Nevertheless, the derived strategies can potentially be used to derive appropriate solutions for solving the conflict. How to automatically achieve strategies that can be directly used as a TA or how to transform a given strategy synthesized by UPPAAL TIGA to a TA is considered to be future work.

6 Compositional Reasoning - Supporting Single I/O Ports

Based on the previously defined property of language progressive/receptive behavior, the compositional reasoning scheme is introduce in this section. It is shown how to build an overall valid (deadlock and zeno free) architecture, based on individual valid (deadlock and zeno free) atomic components while only applying local checks is required.

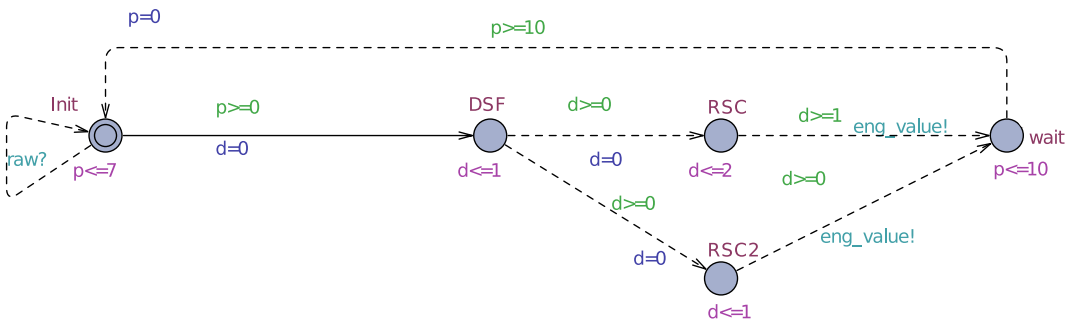


Figure 20: The TIOA $\mathcal{A}_{FS}$ with edges being grouped into controllable and non-controllable.

6.1 Well-Formedness - Language Progressive

Based on the definition of language progressive behavior, a well-formedness criterion for a single component is defined.

Definition 21 (Well-formed Component)
A component with TIOA $\mathcal{A} = (A, I, O, H)$ is well-formed *if $\Upsilon\, pro\, \mathcal{A}$ according to Def. 15 and Υ being the input language of $\mathcal{A}$, only including signals of I.*

A TIOA $\mathcal{A}_\Upsilon$, representing the input language Υ, can be obtained by taking the TIOA associated with the neighbored component that is sending signals to $\mathcal{A}$.

Following two neighbored components are considered while the output of the first is connected to the input of the second. Given TIOA $A_1 = (A_1, I_1, H_1, O_1)$, representing the behavior of the first, and $\mathcal{A}_2 = (A_2, I_2, H_2, O_2)$, representing the behavior of the second component, while both are valid and compatible such that $O_1 = I_2$.

Theorem 6.1 (Composed Components)
The parallel composition of two well-formed compatible components represented by TIOA $\mathcal{A}_1 = (A_1, I_1, O_1, H_1) \parallel \mathcal{A}_2 = (A_2, I_2, O_2, H_2)$ with compatible TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$ is well-formed and valid if $traces(\mathcal{A}_1)\restriction I_2\, pro\, \mathcal{A}_2$.

Proof 6.1
The parallel construct $\mathcal{A}_1 \parallel \mathcal{A}_2$ can only be not well formed if $\neg\Upsilon_1\, pro\, \mathcal{A}_1 \parallel \mathcal{A}_2$. This can only be the case if $\mathcal{A}_2$ blocks at least one transition included in $\mathcal{A}_1$, what is not possible because $traces(\mathcal{A}_1)\restriction I_2\, pro\, \mathcal{A}_2$.

The first result is, that two compatible and well-formed components, sharing either input or output signals, is valid (deadlock and zeno free). Following it is shown that also the property of language progressive behavior is preserved under composition.

Theorem 6.2 (Well-Formed Composition)
Given the parallel composition of two valid, compatible and well-formed components represented by TIOA $\mathcal{A}_1 = (A_1, I_1, O_1, H_1) \parallel \mathcal{A}_2 = (A_2, I_2, O_2, H_2)$ with $O_1 = I_2$ and all other signal sets are disjoint. If $\Upsilon\, pro\, \mathcal{A}_1$ and $\mathcal{A}_1\restriction O_1\, pro\, \mathcal{A}_2$, it follows that $\mathcal{A}_1 \parallel \mathcal{A}_2$ is well-formed for language Υ ($\Upsilon\, pro\, \mathcal{A}_1 \parallel \mathcal{A}_2$).

Proof 6.2
(sketch) Assuming $\neg\Upsilon\, pro\, \mathcal{A}_1 \parallel \mathcal{A}_2$ (not well-formed). Let $\mathcal{T}_1 = (\Sigma_1, \mathcal{S}_1, \mathcal{S}_1^0, T_1)$ be the induced timed system of $\mathcal{A}_1$, $\mathcal{T}_2 = (\Sigma_2, \mathcal{S}_2, \mathcal{S}_2^0, T_2)$ be the induced timed system of $\mathcal{A}_2$ and $\mathcal{T}_{1,2} = (\Sigma_{1,2}, \mathcal{S}_{1,2}, \mathcal{S}_{1,2}^0, T_{1,2})$ be the induced timed system of $\mathcal{A}_1 \parallel \mathcal{A}_2$. It holds that a state $s_{1,2} = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$ exists being a transition blocking state for a trace $\alpha = \beta \circ t \circ \gamma \in \Upsilon$ with $s_1 = \langle l_1, v_1\rangle \in \mathcal{S}_1$, $s_2 = \langle l_2, v_2\rangle \in \mathcal{S}_2$ and $\neg\exists t_2 \in T_{1,2}(s_{1,2})$ such that $t_2\restriction I_1 \vdash t$. Because of $\Upsilon\, pro\, \mathcal{A}_1$ it holds: $\exists t_3 \in T_1(s_1)$ such that $t_3\restriction I_1 \vdash t$. Thus, due to synchronization with $\mathcal{A}_2$ this transition t_3 is no longer observable on $s_{1,2}$. t can be a delay or signal of I_1 only. If t is a delay and in $s_{1,2}$ no transition is observable allowing time to diverge, the induced timed system $\mathcal{T}_2$ of $\mathcal{A}_2$ includes a transition blocking state s_2 for $\mathcal{A}_1\restriction O_1$, $traces(\mathcal{A}_1)\restriction O_1\, pro\, \mathcal{A}_2$ cannot be fulfilled

leading to a contradiction. t can also be a signal included in I_1 that is observable on state s_1 of $\mathcal{A}_1$. Because no signal in I_1 is synchronized with $\mathcal{A}_2$ in $\mathcal{A}_1 \parallel \mathcal{A}_2$, this transition needs to be preserved in state $s_{1,2}$ resp. $T_{1,2}(s_{1,2})$. This cannot be the case and leads to a contradiction.

Theorem 6.2 states that the composition of two well formed components, represented by TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$, while the second consumes signals send by the first, is well-formed for the (input) language Υ of $\mathcal{A}_1$. Following a well-formedness condition for multiple components connected in an acyclic form is defined.

Theorem 6.3 (Multiple Components)
The parallel composition of n well-formed and pair wise compatible components represented by TIOA $\mathcal{A}_1 \parallel ... \parallel \mathcal{A}_n$, for that hold $O_{i-1} = I_i$, with $i \leq n$ and all other sets of signals are disjoint, is well formed and valid if for each $\mathcal{A}_{i-1} = (A_{i-1}, I_{i-1}, O_{i-1}, H_{i-1})$, $\mathcal{A}_i = (A_i, I_i, O_i, H_i)$ with $(i-1, i)$, $i \leq n$ holds $O_{i-1} = I_i$ and $traces(\mathcal{A}_{i-1}) \restriction O_{i-1} \, pro \, \mathcal{A}_i$.

Proof 6.3
(sketch) For $\mathcal{A}_{n-1} \parallel \mathcal{A}_n$, because of Theorem 6.2 holds: if $traces(\mathcal{A}_{n-2}) \restriction_{O_{n-2}} pro \, \mathcal{A}_{n-1}$ and $traces(\mathcal{A}_{n-1}) \restriction O_{n-1} \, pro \, \mathcal{A}_n$, it follows that $traces(\mathcal{A}_{n-2}) \restriction_{O_{n-2}} pro \, \mathcal{A}_{n-1} \parallel \mathcal{A}_n$. By renaming $\mathcal{A}_{n-1} \parallel \mathcal{A}_n = \mathcal{A}'_{n-1}$, it follows inductively for $\mathcal{A}_{n-2} \parallel \mathcal{A}'_{n-1}$ that Theorem 6.3 need to be fulfilled for the overall archtecture.

For the cutout of an overall architecture like shown in Fig. 21 the reasoning scheme allows checking the correctness of the composition by only applying local checks including at most two neighbored components at a time. Thus, instead of being required checking all involved TIOA at once using the parallel product $\mathcal{A}_1 \parallel \mathcal{A}_2 \parallel \mathcal{A}_3 \parallel \mathcal{A}_4$, the reasoning scheme allows checking only pairs of neighbored components of language progressive TIOA.

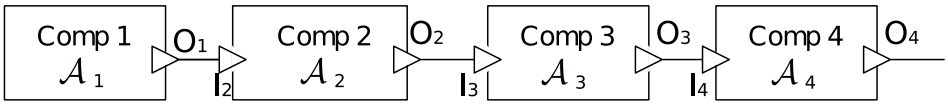


Figure 21: Simplified example of an architecture representing multiple components by TIOA $\mathcal{A}_1, ..., \mathcal{A}_4$.

In summary, the introduced reasoning scheme scales (R1), supports the example introduced in Sec. 3 containing components with restricted resources (R2) and that supports protecting IPs (R3) by only requiring to provide details of neighbored components for realizing the local checks. How requirement (R3) can be further supported is briefly discussed in Sec. 9. As a result, only complex architectures containing multiple ports and cyclic structures (R4) are not sufficiently supported by the introduced approach. In the following the approach is extended such that more complex architectures (R4) containing cyclic dependencies are supported.

6.2 Limitations

Unfortunately, the property of language progressive behavior is not sufficient for supporting cyclic architectures. As an example consider the two TIOA shown in Fig. 22.

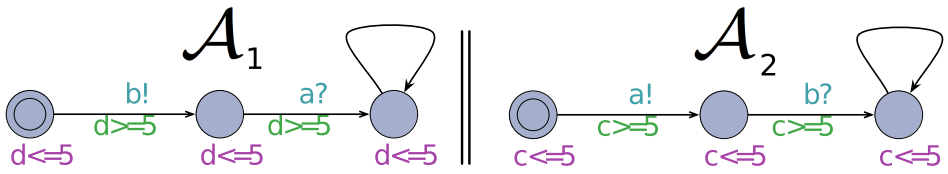


Figure 22: Two compatible TIOA $\mathcal{A}_1 = (A_1, I_1, H_1, O_1)$ and $\mathcal{A}_2 = (A_2, I_2, H_2, O_2)$ with $O_1 = I_2 = \{b\}$, $O_2 = I_1 = \{a\}$ and $H_1 = H_2 = \emptyset$.

Fig. 22 shows an example of two TIOA $\mathcal{A}_1 = (A_1, I_1, H_1, O_1)$ and $\mathcal{A}_2 = (A_2, I_2, H_2, O_2)$ for that hold $traces(\mathcal{A}_1)\restriction O_1 \, pro \, \mathcal{A}_2$ and $traces(\mathcal{A}_2)\restriction_{O_2} pro \, \mathcal{A}_1$. The only trace observable on $\mathcal{A}_1$ is $\alpha_1 = 5 \; b \; 0 \; a \; \infty$ and the only trace observable on $\mathcal{A}_2$ is $\alpha_2 = 5 \; a \; 0 \; b \; \infty$. While for $\Upsilon_1 = traces(\mathcal{A}_1)\restriction O_1 = \{(5 \; b \; \infty)\}$ and $\Upsilon_2 = traces(\mathcal{A}_2)\restriction O_2 = \{(5 \; a \; \infty)\}$ holds $\Upsilon_1 \, pro \, \mathcal{A}_2$ and $\Upsilon_2 \, pro \, \mathcal{A}_1$, for the parallel product $\mathcal{A}_1 \parallel \mathcal{A}_2$ a deadlock obviously exists (signal b cannot be send by $\mathcal{A}_1$ in the initial state and as a result no transition at all can be taken). Thus, an example of two compatible, valid and pairwise language progressive TIOA connected in a cyclic architecture exist that includes a deadlock.

The same holds for the example shown in Fig. 23, where signal a, resp. b cannot be received.

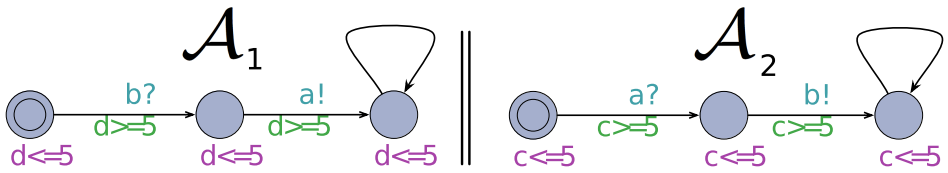


Figure 23: Two compatible TIOA $\mathcal{A}_1 = (A_1, I_1, H_1, O_1)$ and $\mathcal{A}_2 = (A_2, I_2, H_2, O_2)$ with $O_1 = I_2 = \{b\}$, $O_2 = I_1 = \{a\}$ and $H_1 = H_2 = \emptyset$.

In both cases (shown in Fig. 22 and Fig. 23) it holds that each involved TIOA contains a trace where input and output signals are sent and received without time in between. As an example, the TIOA $\mathcal{A}_1$ of Fig. 22 allows observing the trace $\alpha = 5 \; b \; a \; \infty$, where b is an output signal and a is an input signal. Such a trace of a TIOA is defined as follows.

Definition 22 (Zero-Time Input/Output Trace)
A TIOA $\mathcal{A} = (A, I, O, H)$ *contains a* zero-time input/output trace *if* $\exists \alpha \in traces(\mathcal{A})$ *with* $\alpha = \beta \circ \gamma \circ \delta$, $time(\gamma) = 0$, $signals(\gamma) \cap I \neq \emptyset$ *and* $signals(\gamma) \cap O \neq \emptyset$.

Thus, each TIOA containing a *zero-time input/output trace* allows to observe at least one trace where an input signal is received without any delay after sending an output signals or an output signal is send without any delay after receiving an input signal.

Following a condition is given by Theorem 6.4 allowing excluding deadlocks in a cyclic case of language progressive TIOA if for at least one of the TIOA a zero-time input/output trace can be excluded.

Theorem 6.4
For two compatible and valid TIOA $\mathcal{A}_1 = (A_1, I_1, O_1, H_1)$ *and* $\mathcal{A}_2 = (A_2, I_2, O_2, H_2)$ *with* $O_1 = I_2$, $O_2 = I_1$, $\mathcal{A}_1 \restriction O_1 \, pro \, \mathcal{A}_2$ *and* $\mathcal{A}_2 \restriction O_2 \, pro \, \mathcal{A}_1$ *holds: if* $\neg\exists \alpha_1 \in traces(\mathcal{A}_1)$ *being a* zero-time input/output trace *or* $\neg\exists \alpha_2 \in traces(\mathcal{A}_2)$ *being a* zero-time input/output trace*, it follows* $\mathcal{A}_1 \parallel \mathcal{A}_2$ *is deadlock free.*

Proof 6.4
It directly follows from Lemma A.5 (cf. Appendix A) that a deadlock can only occur if both TIOA include such a trace. Thus, if only one TIOA does not include a zero-time input/output trace, no deadlock can occur.

The main result of Theorem 6.4 is, that if at least one TIOA requires time to pass between each pair of send and received signals, no additional deadlock can occur in the cyclic case. The proof of Lemma A.5 has been moved to Appendix A.

Because the composition of an arbitrary number of connected components with at most a single input and output port can be understood as one TIOA, each case can be reduced to a system consisting of two components only, like shown in Fig. 24. In Fig. 24 the two components represented by the TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$ are connected in an acyclic form and become the TIOA $\mathcal{A}_{1||2}$, where signals $O_1 = I_2$ become internal. Thus, according to Theorem 6.4 it is sufficient that component $\mathcal{A}_3$ is language progressive for $\mathcal{A}_{1||2}$, resp. $\mathcal{A}_{1||2}$ is language progressive for $\mathcal{A}_3$, and that $\mathcal{A}_3$ or $\mathcal{A}_{1||2}$ (one is sufficient) contains no zero-time input/output trace. Because for acyclic structures according to Theorem 6.2 the property of language progressiveness is preserved under composition for $\mathcal{A}_{1||2} = \mathcal{A}_1 \parallel \mathcal{A}_2$, it only need to be ensure that zero-time input/output traces can be excluded for $\mathcal{A}_3$ or $\mathcal{A}_{1||2}$ (one is sufficient). Following it is shown how to automatically check if a component excludes zero-time input/output traces.

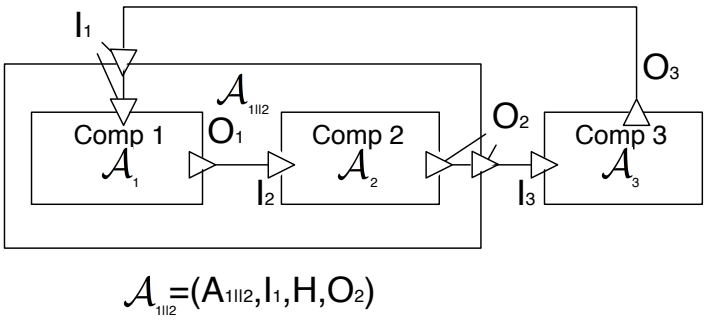


Figure 24: A (cyclic) composition of components can be understood as two components being composed, resulting in a single component.

6.3 Input/Output delayed TIOA

In this section it is discussed, based on the given application example, how to check if a TIOA contains delays between each pair of consecutive send and received signals. If at least one TIOA does not include a zero-time input/output trace, contained in an architecture of components connected in a cyclic form, according to Theorem 6.4 the composition of multiple valid and pairwise language progressive TIOA is valid.

Subsequently it is show how to construct a testbed allowing deciding if a zero-time input/output trace is included in the set of traces observable on a given TIOA $\mathcal{A} = (A, I, O, H)$. For this purpose an initial TIOA $\mathcal{A}_{I,O} = (A_{I,O}, I_{I,O}, O_{I,O}, H_{I,O})$ is created containing five locations $l_0, l_I, l_{IErr}, l_O, l_{OErr}$ and a single clock x that is not contained in $\mathcal{A}$. Further, signal sets are chosen as follows: $I_{I,O} = O$, $O_{I,O} = I$ and $H_{I,O} = \emptyset$. Thus, $\mathcal{A}_{I,O}$ is able to receive signals send by $\mathcal{A}$ and able to send signals that can be received by $\mathcal{A}$.

The construction works as follows: l_0 is the initial location of $\mathcal{A}_{I,O}$ not containing any invariant. For each signal σ in $I_{I,O}$ an edge $e_\sigma = (l_0, \sigma, \emptyset, \{x\}, l_I)$ is added leading to location l_I containing an empty set of guards and resetting the clock x to zero. Thus, from the initial location any signal potentially send by $\mathcal{A}$ can be received in the initial location leading to location l_I while clock x is reset to zero when taking the edge. For each signal $\sigma_2 \in O_{I,O}$ a second edge $e_{\sigma_2} = (l_I, \sigma_2, \varphi, l_{IErr})$, with $\varphi = x \leq 0$ is added. No outgoing transition exists in location l_{IErr} while an invariant $x \leq 0$ is added to this location. Thus, once $\mathcal{A}_{I,O}$ is in location I_{IErr}, a deadlock occurs (no discrete and no delay transitions are possible). One additional edge $e_{I,0} = (l_I, \sigma_\epsilon, \emptyset, \emptyset, l_0)$ is added, not containing any signal, guard or update, leading back to the initial location.

The same is applied for all signals $\sigma \in O_{I,O}$. For each σ an edge $e_\sigma = (l_0, \sigma, \emptyset, \{x\}, l_O)$ is added. Thus, from the initial location any signal potentially received by $\mathcal{A}$ can be send in the initial location while clock x is set to zero when sending the signal. For each signal $\sigma_2 \in I_{I,O}$ a second edge $e_{\sigma_2} = (l_O, \sigma_2, \varphi, l_{OErr})$, with $\varphi = x \leq 0$ is added. No outgoing transition exists in location l_{OErr} while an invariant $x \leq 0$ is added to this location. Thus, once $\mathcal{A}_{I,O}$ is in location I_{OErr}, a deadlock occurs (no discrete and no continuous transitions are possible). One additional edge $e_{I,0} = (l_O, \sigma_\epsilon, \emptyset, \emptyset, l_0)$ is added leading back to the initial location, not containing any signal, guard or update.

By construction holds location l_{OErr} and location l_{IErr} are reachable, iff, at least one input signal is send without any delay after receiving an output signal, resp. an input signal is received without any delay after sending an output signal. Only in location l_{OErr} or location l_{IErr} a state can exists where a signal cannot be send or received for longer than zero time units, necessarily leading to a deadlock.[16] In any other state either all signals can be received, resp. send or a successor state is reachable in zero time (when being in location l_I or l_O) where all signals can be received or send. An example of a testbed, allowing checking if a zero-time input/output trace is observable on a given TIOA, is shown in Fig. 25. It consists of the previously described five locations l_0, l_I, l_O, l_{IErr} and l_{OErr}. Additionally, edges are added for signal *raw* and signal *fuel_rate* according to the previously described procedure.

[16] Time cannot diverge and because each TIOA need to be valid and zeno-free, there need to be a last discrete transition that can be taken if time is not able to diverge.

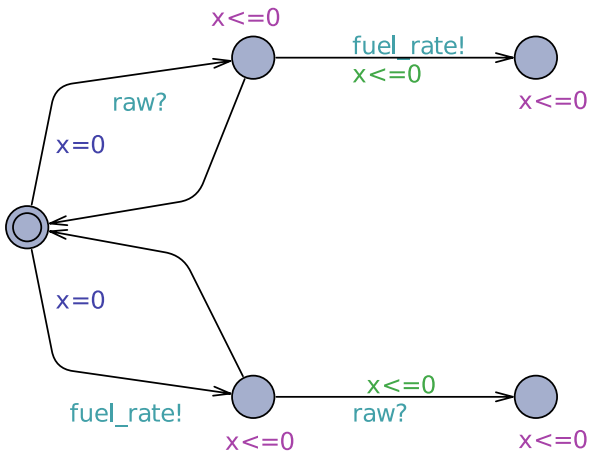


Figure 25: TIOA $\mathcal{A}_{EMIO}$ for checking absence of zero-time input/output traces for the TIOA $\mathcal{A}_{EM}$, representing the behavior of component *EngineModel*.

Fig. 25 shows a testbed allowing deciding if a zero-time input/output trace is observable on the TIOA $\mathcal{A}_{EM}$, representing the behavior of component *EngineModel*. Iff, the TIOA $\mathcal{A}_{EM}$ is able to send and receive signals without any time-delay in between, location l_{IErr} or l_{OErr} are reachable, and a deadlock is reachable in the induced timed-system of the parallel product $\mathcal{A}_{EMIO} \parallel \mathcal{A}_{EM}$.

7 Compositional Reasoning - Supporting Multiple I/O Ports

Based on the previously given definition of language progressive behavior an overall reasoning scheme for architectures with arbitrary structures is defined. First, the reasoning scheme is extended for supporting components containing multiple input and output ports. Second, it is shown how to support cyclic connections using components with multiple ports. By doing so, a reasoning scheme is defined supporting all requirements (R1)...(R4).

7.1 Well-Formedness - Supporting Multiport Components

Till now a reasoning scheme has been defined supporting components represented by TIOA where each component has one input and/or one output port only. Following the reasoning scheme is extended to support components with multiple ports like included in the example shown in Fig. 1.

The formal description of a component with n different input and m different output ports, represented by the TIOA $\mathcal{A} = (A, I, O, H)$, is realized by partitioning the input signals I

respectively the output signals O. The partitioning of input signals results in different sets $I_1, ..., I_n$ with $I_1 \cap ... \cap I_n = \emptyset$ such that $I_1 \cup ... \cup I_n = I$. Partitioning of output signals O results in sets $O_1, ..., O_m$ with $O_1 \cap ... \cap O_m = \emptyset$ such that $O_1 \cup ... \cup O_m = O$. The language of the input port i is defined by Υ_i consisting only of signals in I_i. Accordingly the language for the output port j is defined by the Θ_j consisting only of signals in O_j. By construction via the partitioning of signals it is ensured that $\Upsilon_1 \cap ... \cap \Upsilon_n = \emptyset$ and $\Theta_1 \cap ... \cap \Theta_m = \emptyset$.

Given a TIOA $\mathcal{A} = (A, I, O, H)$ and the set of input ports $I_1, ..., I_n$ associated with the related languages in form of trace sets $\Upsilon_1, ..., \Upsilon_n$, and the set of output ports $O_1, ..., O_m$ associated with the languages $\Theta_1, ..., \Theta_m$. It is required for a component to be well formed that the property of language progressive behavior is fulfilled for all traces resulting from the interleaving of the languages associated with the input ports. The interleaving of languages is defined in form of the product language according to Def. 23.

Definition 23 (Product language)
For a finite set of signals Σ, let Σ^ω denote the set of all finite and infinite traces over Σ. Given two languages Υ_1 and Υ_2 with $\Sigma_1 = signals(\Upsilon_1)$ and $\Sigma_2 = signals(\Upsilon_2)$ such that $\Sigma_1 \cap \Sigma_2 = \emptyset$. The product language *over Υ_1 and Υ_2 is defined by:*

$$\Upsilon_1 \times \Upsilon_2 = \left\{ \alpha \in \left(\Sigma_1 \cup \Sigma_2 \cup \mathbb{R}^+\right)^\omega \mid \alpha{\restriction}\Sigma_1 \in \Upsilon_1 \text{ and } \alpha{\restriction}\Sigma_2 \in \Upsilon_2 \right\}$$

As an example the product language of the two languages $\Upsilon_1 = \{\alpha_1\}$ and $\Upsilon_2 = \{\alpha_2\}$ with $\alpha_1 = 1\ a\ \infty$ and $\alpha_2 = 1\ b\ \infty$ is $\{\{1\ a\ b\ \infty\}, \{1\ b\ a\ \infty\}\}$, where ∞ is the progress of time against infinity.

Consider the parallel product $\mathcal{A}_1 \parallel \mathcal{A}_2$ of the two valid TIOA $\mathcal{A}_1 = (A_1, I_1, H_1, O_1)$ and $\mathcal{A}_2 = (A_2, I_2, H_2, O_2)$ with disjoint signals and variables (clocks and attributes). In other words both TIOA are independent because they do not share any syntactical elements that need to synchronize. Because synchronization does not occur and no deadlock or zeno behavior can exist (because both are valid), the observable traces $traces(\mathcal{A}_1 \parallel \mathcal{A}_2)$ are equal to the product language of the individual trace sets $traces(\mathcal{A}_1) \times traces(\mathcal{A}_2)$.

Theorem 7.1
The language $\Upsilon_{1,2}$ of the parallel product of two valid TIOA $\mathcal{A}_1 = (A_1, I_1, H_1, O_1)$ and $\mathcal{A}_2 = (A_2, I_2, H_2, O_2)$ with $O_1 \cap I_2 = O_2 \cap I_1 = \emptyset$ that do not share any variable is $\Upsilon_{1,2} = traces(\mathcal{A}_1) \times traces(\mathcal{A}_2)$.

Proof 7.1
(sketch) The observable traces of the parallel product do not result in the product language, iff, synchronization prevents at least one trace or a state is reachable in one of the TIOA where progress of time is not possible. Because the TIOA do not share any syntactical element, synchronization effects cannot take place. Because only valid (deadlock and zeno free) TIOA are considered, time is always able to diverge. Thus, no trace of a TIOA can be prevented by a zeno or deadlock trace included in any TIOA. As a result Theorem 7.1 is fulfilled.

In the reminder of this work such TIOA are called to be independent if no syntactical elements in form of signals and variables are shared. This property of the product language for

independent and valid TIOA is subsequently used for being able to define a well formedness criterion for components with multiple input ports connected to different components.

Definition 24 (Multiport Component)
A multiport component C *represented by a valid TIOA* $\mathcal{A} = (A, I, O, H)$ *contains a partitioning* $\mathcal{IP} = \{I_1, ..., I_n\}$ *of pairwise disjoint input signal sets and a partitioning* $\mathcal{OP} = \{O_1, \ldots, O_m\}$ *of pairwise disjoint output signal sets. It further holds that* $I = I_1 \cup ... \cup I_n$ *and* $O = O_1 \cup ... \cup O_m$*. For each* $I_i \in \mathcal{IP}$ *a language* Υ_i *exists containing only signals included in* I_i *and for each* $O_j \in \mathcal{OP}$ *a language* Θ_j *exists containing only signals included in* O_j*.*

Following a well-formed property for a *multiport component* using the notion of the product language is defined.

Definition 25 (Well-Formed Multiport Component)
A component represented by a valid TIOA $\mathcal{A} = (A, I, O, H)$ *with multiple input ports* $I_1, ..., I_n$*, associated with the related languages in form of trace sets* $\Upsilon_1, ..., \Upsilon_n$ *and* $I_i \cap I_j = \emptyset$ *with* $i \neq j$ *and* $1 \leq i, j \leq n$ *is well-formed, if* $\Upsilon_1 \times ... \times \Upsilon_n \, pro \, \mathcal{A}$*.*

A *link* between an output port of a first component, represented by a TIOA $\mathcal{A}_1$, and an input port of a second component, represented by $\mathcal{A}_2$ exists, iff, the associated ports are connected. A *link* is defined as follows.

Definition 26 (Link)
A link ln *between two different components, represented by two different TIOA* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *is a tuple* $ln = \langle O_i, I_j \rangle$ *with* O_i *being an output port of* $\mathcal{A}_1$ *and* I_j *being an input port of TIOA* $\mathcal{A}_2$*.*

Between components *FuelSensor* and *FuelController* (see Fig. 1) two links according to Def. 26 exist in case of the connected input, resp. output port with name *mode* and the connected input, resp. output port with name *correctedSensors*. When connected via a link the input port receives each signal send by the output port. Following the definition of a well-formed link that connects components is given. Each component is identified via its name (e.g., component with name *FuelController* like shown in Fig. 1).

Definition 27 (Well-Formed Link)
Given a link $ln = \langle O_i, I_j \rangle$ between output port i providing the output language Θ_i of a first well-formed component with name C_1, and an input port j providing the input language Υ_j of a second well-formed component with name C_2, with $C_1 \neq C_2$. Let the two compatible TIOA $\mathcal{A}_1 = (A_1, I_1, O_1, H_1)$ and $\mathcal{A}_2 = (A_2, I_2, O_2, H_2)$ represent component C_1, resp., component C_2. This link is well-formed if

$$\Theta_{1i} \subseteq \Upsilon_{2j}.$$

The composition of the two components C_1 and C_2 is well-formed if all links between output ports of the first and input ports of the second are well-formed.

7.2 Extended Scheme

The main result for the composition of components employing the concept of language progressive behavior, is, that well-formed components and well-formed compositions imply that the resulting composed component and its timed automata are also well-formed if connect in an acyclic form.

Theorem 7.2
Given a composition of two well-formed multiport components with TIOA $\mathcal{A}_1 = (A_1, I_1, O_1, H_1)$ and $\mathcal{A}_2 = (A_2, I_2, O_2, H_2)$, well-formed links $L = \{ln_1, ..., ln_n\}$ and for each link $ln_k = \langle O_i, I_j \rangle$ holds O_i is an output port of $\mathcal{A}_1$ and I_j is an input port of $\mathcal{A}_2$ (no links exist building a cyclic connection). The composition $\mathcal{A}_3 = \mathcal{A}_1 \| \mathcal{A}_2$, with the resulting set of input and output ports for $\mathcal{A}_3$ is well-formed.

Proof 7.2
(sketch) Let $\mathcal{A}_3 = (A_1 \parallel \mathcal{A}_2, I_3, O_3, H_3)$ be the TIOA $\mathcal{A}_3 = \mathcal{A}_1 \parallel \mathcal{A}_2$ with a resulting set of n unconnected (not linked) input ports $I_{3,1}, \ldots, I_{3,n}$ and there associated languages $\Upsilon_{3,1}, \ldots, \Upsilon_{3,n}$. Let $I_{3,1} \cup \cdots \cup I_{3,n} = I_3^{\cup}$ be the union over the resulting input signals of $\mathcal{A}_3$ and let $\Upsilon_{3,1} \times \cdots \times \Upsilon_{3,n} = \Upsilon_3^{\times}$ be the product language of the remaining unconnected input ports of $\mathcal{A}_3$.

$\mathcal{A}_3$ is not well-formed if a transition blocking state s_b according to Def. 16 exists in the induced timed system $\mathcal{T}_3 = (\Sigma_3, \mathcal{S}_3, \mathcal{S}_3^0, T_3)$ of $\mathcal{A}_3$, reachable via a trace β with $\beta \in traces\_to(s_b, \mathcal{A}_3) \restriction I_3^{\cup}$, $\alpha = \beta \circ t \circ \gamma$, $\alpha \in \Upsilon_3^{\times}$ and $\neg \exists t_b \in T_3(s_b)$ with $t_b \restriction I_3^{\cup} \vdash t$. For s_b written as $s_b = \langle \langle l_1, l_2 \rangle, \langle v_1, v_2 \rangle \rangle$ holds that $s_1 = \langle l_1, v_1 \rangle$ is the state information of $\mathcal{A}_1$ in state s_b and $s_2 = \langle l_2, v_2 \rangle$ is the state information of $\mathcal{A}_2$ in s_b.

Assuming a transition blocking state s_b exists. Two different cases can exist: Either 1) t is a delay, not including a signal or 2) t is a signal.

Assuming 1): Because $\Upsilon_1^{\times}\, pro\, \mathcal{A}_1$ it holds that a transition t_1 need to exist in $T_1(s_1)$ of the induced timed system of $\mathcal{A}_1$, such that $t_1 \restriction I_1^{\cup} \vdash t$. If s_1 in s_b is a transition blocking state for $\mathcal{A}_3 = \mathcal{A}_1 \parallel \mathcal{A}_2$, it also holds that t_1 is blocked in $\mathcal{A}_1 \parallel \mathcal{A}_2$ via synchronization with $\mathcal{A}_2$. t_1 cannot contain an input signal of $\mathcal{A}_1$ (than t cannot be blocked via synchronization with $\mathcal{A}_2$), neither contain an internal signal or the empty signal σ_ϵ (both types of transitions do not synchronize with $\mathcal{A}_2$ and cannot be blocked). As a consequence it holds that t_1 is either a

delay transition or a discrete transition including an input signal of $\mathcal{A}_2$. s_1 need to be a state where $T_1(s_1)$ includes a transition (transition t_1) that is blocked by $\mathcal{A}_2$ via synchronization when $\mathcal{A}_2$ is in state s_2 of the transition blocking state s_b. Because of the well-formedness of the links it holds that state s_2 of $\mathcal{A}_2$ is reachable via a trace α_2 for that holds $\alpha_2 \restriction I_2^{\cup} = \beta_2 \circ t_1 \circ \delta_2$, $\alpha_2 \in \Upsilon_2^{\times}$ and $\neg\exists t_2 \in T_2(s_2)$ with $t_2 \vdash t_1$. This is directly in contradiction to $\Upsilon_2^{\times}\, pro\, \mathcal{A}_2$.

Assuming 2): If t is a signal again two different cases can occur. 2.1) t is a signal included in an input port of $\mathcal{A}_2$ and 2.2) t is a signal included in an input port of $\mathcal{A}_1$. Assuming 2.1): Because of the well-formedness of links it follows that s_2 of $\mathcal{A}_2$ is reachable via a trace β_2 with $\alpha_2 = \beta_2 \circ t \circ \delta_2$, $\alpha_2 \in \Upsilon_2^{\times}$ and there not exists a transition $t_2 \in T_2(s_2)$ such that $t_2 \restriction I_2^{\cup} \vdash t$. In this case s_2 itself need to be a transition blocking state and $\Upsilon_2^{\times}\, pro\, \mathcal{A}_2$ cannot be fulfilled, leading to a contradiction.

Assuming 2.2): Because of $\Upsilon_1^{\times}\, pro\, \mathcal{A}_1$ it holds that a transition t_1 exists in $T_1(s_1)$, such that $t_1 \restriction I_1^{\cup} \vdash t$. If s_1 in s_b is a transition blocking state for $\mathcal{A}_3 = \mathcal{A}_1 \parallel \mathcal{A}_2$, it holds that t_1 is blocked in $\mathcal{A}_1 \parallel \mathcal{A}_2$ via synchronization with $\mathcal{A}_2$. t_1 cannot contain a continuous transition or a transition including an input signal of $\mathcal{A}_1$ (in both cases follows $\neg t_1 \vdash t$), neither a transition including an internal signal nor the empty signal σ_ϵ (in both cases this transition won't be removed due to synchronization with $\mathcal{A}_2$). Thus, t_1 need to be a transition including an input signal of $\mathcal{A}_2$. s_1 need to be a state where $T_1(s_1)$ includes a transition (transition t_1) that is blocked by $\mathcal{A}_2$ via synchronization when $\mathcal{A}_2$ is in state s_2 of the transition blocking state s_b. Because of the well-formedness of the links it holds that state s_2 of $\mathcal{A}_2$ is reachable via a trace α_2 for that holds $\alpha_2 \restriction I_2^{\cup} = \beta_2 \circ t_1 \circ \delta_2$, $\alpha_2 \in \Upsilon_2^{\times}$ and $\neg\exists t_2 \in T_2(s_2)$ with $t_2 \vdash t_1$. This is directly in contradiction to $\Upsilon_2^{\times}\, pro\, \mathcal{A}_2$.

It follows that for all possible cases no transition blocking state can exist and Theorem 7.2 need to be fulfilled.

Thus, for two well-formed multiport components the property of language progressive behavior is preserved under composition, if connected in an acyclic way using well-formed links only.

For being able to extend the scheme to support cyclic connections of links for multiport components, the semantics of a cyclic dependency is defined.

Definition 28 (Cyclic Dependency)
Given a set of n well-formed components $C_1, \ldots, C_n = C$ connected via a set L of well-formed links. A Cyclic dependency *exists if the following conditions hold: a link $l_i = \langle O_l, I_m \rangle \in L$ connects an output port of Component $C_q \in C$ to an input port of component $C_r \in C$. A set of links $L_r \subset L$ exists with $L_r \cap l_i = \emptyset$, and L_r builds a connected graph of components $\{C_a, \ldots, C_b\} \subset C$ with $C_q \notin \{C_a, \ldots, C_b\}$ and $C_r \in \{C_a, \ldots, C_b\}$. A link $l_z = \langle O_x, I_z \rangle \in L$ exists where an output port of a component in $\{C_a, \ldots, C_b\}$ is linked to an input port of C_q.*

Subsequently it is shown that the property of language progressive behavior is preserved under composition containing cyclic dependencies if at least one of the composed components excludes zero-time input/output traces.

Theorem 7.3 (Well-Formed Cyclic Multiport)
Given a composition of two well-formed multiport components with TIOA $\mathcal{A}_1 = (A_1, I_1, O_1, H_1)$ and $\mathcal{A}_2 = (A_2, I_2, O_2, H_2)$, well-formed links $L = \{ln_1, ..., ln_n\}$ and for each link $ln_k = \langle O_i, I_j \rangle$ holds O_i is an output port and I_j is an input port (cyclic dependencies according to Def. 28 are allowed). The composition $\mathcal{A}_3 = \mathcal{A}_1 \| \mathcal{A}_2$, with the resulting set of input and output ports for $\mathcal{A}_3$ is well-formed if $\mathcal{A}_1$ or $\mathcal{A}_2$ not contains a zero-time input/output trace according to Def. 22.

Proof 7.3
(sketch) It follows directly from Theorem 7.2 that in the case no cyclic dependency of links exists, the property of language progressive behavior needs to be fulfilled for $\mathcal{A}_3$. Following it is shown that in the case a cyclic connection of links is included, both TIOA need to contain a zero-time input/output trace if the property of language progressive behavior is violated for the product language of the remaining unconnected ports. Thus, if at least one TIOA does not include a zero-time input/output trace, the property of language progressive behavior needs to be fulfilled.

Let $\mathcal{A}_3 = (A_1 \parallel \mathcal{A}_2, I_3, O_3, H_3)$ be the TIOA $\mathcal{A}_3 = \mathcal{A}_1 \parallel \mathcal{A}_2$ with a resulting set of n unconnected (not linked) input ports $I_{3,1}, \ldots, I_{3,n}$ and there associated languages $\Upsilon_{3,1}, \ldots, \Upsilon_{3,n}$. Let $I_{3,1} \cup \cdots \cup I_{3,n} = I_3^{\cup}$ be the union over the resulting input signals of $\mathcal{A}_3$ contained in unconnected input ports and let $\Upsilon_{3,1} \times \cdots \times \Upsilon_{3,n} = \Upsilon_3^{\times}$ be the product language of the remaining unconnected input ports of $\mathcal{A}_3$. Let $\Upsilon_{1,2}^{\times}$ describe the product language of all languages associated with input ports (linked or not) of $\mathcal{A}_1$ and $\mathcal{A}_2$.

If $\mathcal{A}_3$ is not well-formed a transition blocking state s_b according to Def. 16 exists in the induced timed system $\mathcal{T}_3 = (\Sigma_3, \mathcal{S}_3, \mathcal{S}_3^0, T_3)$ of $\mathcal{A}_3$ reachable via a trace $\beta_{\cup}$ with $\beta_{\cup} \in traces\_to(s_b, \mathcal{A}_3) \restriction I_3^{\cup}$, $\alpha_{\cup} = \beta_{\cup} \circ t \circ \gamma_{\cup}$, $\alpha_{\cup} \in \Upsilon_3^{\times}$ and $\neg \exists t_b \in T_3(s_b)$ with $t_b \restriction I_3^{\cup} \vdash t$.

Because $\alpha_{\cup} \in \Upsilon_3^{\times}$ there need to exist a trace $\alpha \in \Upsilon_{1,2}^{\times}$, the product language over all input ports of both TIOA, with $\alpha = \beta \circ t \circ \gamma$, $\beta \in traces\_to(s_b, \mathcal{A}_3)$ and $\neg \exists t_2 \in T_3(s_b)$ with $t_2 \restriction I_3^{\cup} \vdash t$. For s_b written as $s_b = \langle \langle l_1, l_2 \rangle, \langle v_1, v_2 \rangle \rangle$ need to hold that $s_1 = \langle l_1, v_1 \rangle$ is the state information of $\mathcal{A}_1$ in state s_b and $s_2 = \langle l_2, v_2 \rangle$ is the state information of $\mathcal{A}_2$ in s_b.

Assuming the parallel composition is not well-formed, two different cases can exist for the trace α and the state s_b: 1) for s_b with $\alpha = \beta \circ t \circ \gamma$ holds t is a signal contained in an input port of $\mathcal{A}_1$, resp. of $\mathcal{A}_2$ or 2) t is a delay containing no signal.

Before investigating the individual cases a partitioning of the links is defined: L_1 is defined as the set of well-formed links connecting output ports of $\mathcal{A}_1$ with input ports of $\mathcal{A}_2$ and L_2 is defined as the set of well-formed links connecting output ports of $\mathcal{A}_2$ with input ports of $\mathcal{A}_1$. By construction it holds that $L_1 \cap L_2 = \emptyset$ and $L_1 \cup L_2 = L$. Because of Theorem 7.2 it further need to hold that the composition of the two well-formed multiport components represented by TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$, using either the set of well-formed links L_1 or the set L_2, need to be well-formed. This is the case because no cyclic dependency exists when using either link set L_1 or link set L_2.

Assuming case 1): Because for $\mathcal{A}_1 \parallel \mathcal{A}_2 = \mathcal{A}_{3,1}$ using the set of well-formed links L_1 only, according to Theorem 7.2 it needs to hold that $\mathcal{A}_{3,1}$ is well-formed (no cyclic connection is included). According to the assumption it further need to hold that $\mathcal{A}_3$, including the set

of well-formed links $L_1 \cup L_2 = L$, is not well formed and s_b is a transition blocking state for the trace $\alpha = \beta \circ t \circ \gamma$ while s_b is no transition blocking state for $\mathcal{A}_{3,1}$. Thus, due to synchronization with signals contained in the set of linked ports of L_2 a transition included in s_b need to be removed by synchronizing a signal of O_2. As a consequence there need to exist a transition $t_{3,1} \in T_{3,1}(s_b)$ of the induced timed system $\mathcal{T}_{3,1} = (\Sigma_{3,1}, \mathcal{S}_{3,1}, \mathcal{S}^0_{3,1}, T_{3,1})$ of $\mathcal{A}_{3,1}$ that is a discrete transition (it cannot be a continuous transition because any continuous transition does not contribute to t in the considered case) including a signal of a port that is also included in one of the links in L_2 and as a consequence including a signal of O_2. $t_{3,1}$ cannot be a transition receiving a signal that becomes enabled when removing all links in L_2, because either such a received signal is equal to t and s_b won't be a transition blocking state for $\mathcal{A}_3$, or it is different to t and as a consequence does not contribute to α because such a transition includes a signal of the input, not contributing to α. As a consequence it needs to be a send signal. Because in L_2 only signals are included that are send by $\mathcal{A}_2$, it needs to be an output signal σ_2 that can be observed on a transition t_2 included in $T_2(s_2)$, the part of the state information of $\mathcal{A}_2$ in state s_b.

The same need to hold for $\mathcal{A}_1 \parallel \mathcal{A}_2 = \mathcal{A}_{3,2}$, using the set of well-formed links L_2 only, for the state s_1 of the induced timed system of $\mathcal{A}_1$. $T_1(s_1)$ need to contain a transition t_1 including a send signal $\sigma_1 \in O_1$ for that holds it is also included in O_1.

Assuming case 2): Because for $\mathcal{A}_1 \parallel \mathcal{A}_2 = \mathcal{A}_{3,1}$, using the set of well-formed links L_1 only, according to Theorem 7.2 it needs to hold that $\mathcal{A}_{3,1}$ is well-formed. According to the assumption it further need to hold that $\mathcal{A}_3$, with the set of well-formed links $L_1 \cup L_2 = L$, is not well formed and s_b is a transition blocking state for the trace $\alpha = \beta \circ t \circ \gamma$ while s_b is no transition blocking state for $\mathcal{A}_{3,1}$. Thus, due to synchronization with signals contained in the set of linked ports of L_2 a transition included in s_b need to be removed. As a consequence there need to exist a transition $t_{3,1}$ in $T_{3,1}(s_b)$ of the induced timed system $\mathcal{T}_{3,1} = (\Sigma_{3,1}, \mathcal{S}_{3,1}, \mathcal{S}^0_{3,1}, T_{3,1})$ of $\mathcal{A}_{3,1}$ that is a discrete transition (a continuous transition won't be removed in this case by synchronizing the signals) including a signal of a port that is included in one of the links in L_2. It needs to be a transition sending a signal, because in any other case (if it is a signal being received) it will be different to t, not contributing to α, and $\mathcal{A}_{3,1}$ won't be language progressive for its input languages also if links L_2 are removed. Thus, it needs to be a signal σ_2 send by a transition $t_2 \in T_2(s_2)$ of $\mathcal{A}_2$.

The same need to hold in case $\mathcal{A}_1 \parallel \mathcal{A}_2 = \mathcal{A}_{3,2}$ is considered using the set of well-formed links L_2 only for the state s_1 of the induced timed system of $\mathcal{A}_1$. $T_1(s_1)$ need to contain a transition t_1 including a send signal $\sigma_1 \in O_1$ for that holds it is also included in I_2.

As a consequence the following conditions need to hold if the property of language progressive behavior is violated for the cyclic composition of two well-formed components C_1 and C_2, represented by the TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$, that are connected via well-formed links L only:

According to the assumption a transition blocking state $s_b = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$, with $s_1 = \langle l_1, v_1\rangle$ representing the state information of $\mathcal{A}_1$ and $s_2 = \langle l_2, v_2\rangle$ representing the state information of $\mathcal{A}_2$ need to exist. Like previously shown it further holds:

$\exists t_1 \in T_1(s_1)$ with $t_1 = s_1 \xrightarrow{\sigma_1} s_1'$, $\sigma_1 \in O_1 \cap I_2$ and $\exists t_2 \in T_2(s_2)$ with $s_2 \xrightarrow{\sigma_2} s_2'$, $\sigma_2 \in O_2 \cap I_1$.

When using the set of links L_1 only, in state s_2 of the parallel product $\mathcal{A}_1 \parallel \mathcal{A}_2$, a transition $t_2 = s_2 \xrightarrow{\sigma_2} s_2'$ exists (because signals of O_2 do not need to be synchronized when using only links L_1) leading to state $s_b' = \langle\langle l_1, l_2'\rangle, \langle v_1, v_2'\rangle\rangle$ with $s_2' = \langle l_2', v_2'\rangle$. Because $t_1 \in O_1 \cap I_2$, $t_1 = s_1 \xrightarrow{\sigma_1} s_1' \in T_1(s_1)$ and because of the langauge progressiveness for $\mathcal{A}_1 \parallel \mathcal{A}_2$ using links L_1, a successor state $s_2'^n$ of s_2 need to be reachable in zero time, allowing receiving signal σ_1. Thus, for $\mathcal{A}_2$ a zero-time input/output trace can be constructed as follows: The zero-time input/output trace starts at state s_2 allowing observing signal σ_2 being an output signal of $\mathcal{A}_2$. The trace is continued leading to state $s_2'^n$ without consuming time. In this state the input signal σ_1 need to be observable because $\mathcal{A}_2$ is language progressive for the trace $\beta \circ t_1$ where $\beta \in traces\_to(s_2'^n, \mathcal{A}_2)$.

A zero-time input output trace for $\mathcal{A}_1$ can be constructed in the same way. As a consequence Theorem 7.3 need to be fulfilled.

Furthermore, the construction of well-formedness results in a situation that a closed component based architecture excludes undesired behavior like deadlock states or zeno traces, if it has been composed step-wise using well-formed atomic components where for each composition of two well-formed atomic components including a cyclic dependency, at least one excludes zero-time input/output traces.

Definition 29 (Well-Formed Composition)
Given two well-formed multiport components C_1 and C_2, represented by TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$, connected via well-formed links L only. Well call $C_1 \parallel C_2 = C_{1,2}$ a well-formed composition *if no cyclic dependency for C_1 and C_2 according to Def. 28 exists, or, if a cyclic dependency exists, $\mathcal{A}_1$ or $\mathcal{A}_2$ excludes zero-time input/output traces according to Def. 22.*

Corollary 7.1
A closed component with timed input/output automaton $\mathcal{A} = (A, \emptyset, \emptyset, H)$ that is a result of a well-formed composition (cf. Def. 29) of two well-formed components is deadlock free and zeno free.

Proof 7.4
(sketch) Assuming that it contains undesired behavior, then, either (i) zeno behavior or (ii) a deadlock is included in one of the components. In case (i) the projection of the zeno trace would imply also that a zeno trace existed for one of the components. In general this cannot be the case for the used model of TA. For case (ii) holds that neither there could be a local deadlock in one of the components nor can synchronization cause a deadlock due to the language progressiveness (+ absence of zero-time input/output traces in case of a cyclic dependency) of the involved automata. Thus, there could not be any undesired behavior in the composed component.

The Def. 29 together with the Corollary 7.1 enable to compositionally check that arbitrary complex architectures will in the end compose to a free system excluding undesired behavior in form of deadlock states or zeno traces. What is only required is

- Ensure that all employed atomic components are well-formed by showing that they are language progressive for the languages associated with the input ports.
- In case of a cyclic dependency according to Def. 29, ensure that one of the two TIOA involved in each check excludes zero-time input/output traces.

8 Checking Well-Formedness on Example

In this section it is shown how to apply the previously introduced reasoning scheme on the application example described in Sec. 3.

8.1 Checking Language Progressive Behavior - Application Example

Following it is shown what is required to ensure that the overall composition of the Fuel-Control-System, shown in Fig. 1 (resp. Fig. 26), is well-formed and as a consequence also deadlock and zeno free. This is achieved by showing that all links between atomic components are well-formed in such a manner that they are language progressive for the languages associated with their linked input ports according to Def. 25. Given the TIOA $\mathcal{A}_{FC}$ representing the atomic component *FuelController* of Fig. 1. The input languages $\Upsilon_{2,1}$ and $\Upsilon_{2,2}$ do represent the input language of port $I_{2,1}$ (associated with port *correctedSensors*) and port $I_{2,2}$ (associated with port *mode*). According to Def. 25 component *FuelController* is well-formed for the two languages associated with the linked input ports, iff: $\Upsilon_{2,1} \times \Upsilon_{2,2}\, pro\, \mathcal{A}_{FC}$. In this particular case both input ports are linked to the output ports of the same component *FuelSensor*. As a results both ports can be seen as a single one and as a consequence TIOA $\mathcal{A}_{FS}$ is taken for representing both input languages $\Upsilon_{2,1}$ and $\Upsilon_{2,2}$. Accordingly well-formedness of the linked ports is fulfilled if $traces(\mathcal{A}_{FS})\, pro\, \mathcal{A}_{FC}$.

The first part of the testbed $TB^{s}_{FC}(\mathcal{A}_{FS}, \mathcal{A}_{FC})$ is shown in Fig. 31, representing the modified TIOA $\mathcal{A}^{c}_{FS}$ of component *FuelSensor*. Modifications are applied according to Sec. 5.2, such that the modified TIOA can be used for checking if the property of language progressive behavior is fulfilled. The second part of the testbed is the TIOA shown in Fig. 7, representing the original behavior of the component *FuelController*.

The same is applied for the two components *EngineModel* and *FuelSensor* like already described in Sec. 5.2, as well as for the two components *FuelController* and *EngineModel*. The first part of testbed $TB_{EM}(\mathcal{A}_{FC}, \mathcal{A}_{EM})$, representing the modified version of the TIOA representing the behavior of component *FuelController*, is shown in Fig. 32. As a result the three testbeds $TB^{s}_{FC}(\mathcal{A}_{FS}, \mathcal{A}_{FC})$, $TB_{EM}(\mathcal{A}_{FC}, \mathcal{A}_{EM})$ and $TB_{FS}(\mathcal{A}_{EM}, \mathcal{A}_{FS})$ have been checked for ensuring the well-formedness of the overall composition. Because a cyclic dependency

according to Def. 28 exists in the architecture shown in Fig. 26, it needs to be ensured that for at least one of the involved components zero-time input/output traces can be excluded. This can be checked by building the parallel product $\mathcal{A}_{EM} \parallel \mathcal{A}_{EMIO}$ of the TIOA $\mathcal{A}_{EM}$ and the created testbed $\mathcal{A}_{EMIO}$ like shown in Fig. 25.

A schematic description of the used testbeds for verifying the well-formedness for the involved TIOA is shown in Fig. 26.[17]

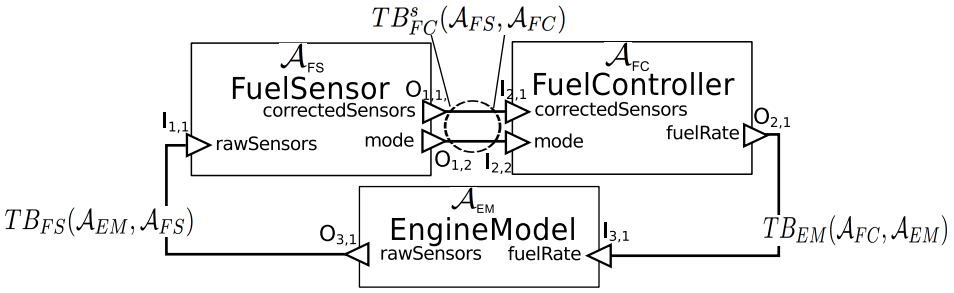


Figure 26: Schematic description of the testbeds associated with the links between the involved components, resp. between the associated TIOA.

It turned out that the TIOA $\mathcal{A}_{FS}$, representing the behavior of component *EngineModel*, is language progressive for the output language defined by the TIOA $\mathcal{A}_{FC}$, representing the timing behavior of component *FuelController*. Like previously shown in Sec. 5.2 a conflicting state s_1 exists in case of component *FuelSensor*, discovered using the testbed $TB(\mathcal{A}_{EM}, \mathcal{A}_{FS})$. As a result one conflicting (not well-formed) link is discovered in the application example indicated by the discovered deadlock state s_1. This conflicting link exists between components *EngineModel* and *FuelSensor*. Like discussed in Sec. 8.3 the proposed reasoning scheme has been applied on the application example and the result have been compared with a monolithic approach where all involved TIOA are analyzed at once. It turned out that also for the parallel construct of all involved TIOA state s_1 is a deadlock state. For our application example it turned out that all discovered violations of language progressive behavior also lead to undesired behavior for the parallel product of the overall architecture. The case discovered when applying the modular approach turns out to be not a false-negative.

For any architecture including an arbitrary number of components, within each testbed at most the behavior models of those components are included that are linked to input ports. As a result requirement (R1) is fulfilled and do to the fact that cyclic dependencies and multiple ports are allowed requirement (R4) is supported. Because (R2) is fulfilled by using the property of language progressive instead of progressive behavior only requirement (R3) till now is not discussed in more detail. In Sec. 9 it is discussed how to further support requirement (R3): protecting IPs.

[17] It is sufficient checking only one connecting component closing the cyclic architecture for zero-time input/output traces. For the rest of the architecture it is sufficient to only check language progressive behavior. This is the case because multiple language progressive TIOA ($\mathcal{A}_{FS}$ and $\mathcal{A}_{FC}$) connected in an acyclic way can be understood as one composition ($\mathcal{A}_{FS} \parallel \mathcal{A}_{FC}$), and according to Theorem 7.3 a deadlock can only occur, if both involved TIOA ($\mathcal{A}_{FS} \parallel \mathcal{A}_{FC}$ and $\mathcal{A}_{EM}$) include zero-time input/output trace.

8.2 Checking Language Receptive Behavior - Application Example

Checking if all involved TIOA representing the behavior of the individual components are language progressive for the linked input languages of their input ports is the first step. When applying the previously described checks using the testbed, potentially one of the required well-formedness criteria is violated. In our application example one conflict has been detected like previously shown in case of state s_1 of component *FuelSensor* (compare Sec. 5.2 and Sec. 8.1). Like discussed in Sec. 5.3 state s_1 can be eliminated by choosing an appropriate valid strategy.

On the one hand derived strategies do well fit to embedded components due to the fact that only edges representing internal behavior parts, in the case of edges including internal signals, are used to realize strategies. Such internal signals are often related to parameters being modified during configuration activities. On the other hand exchanging the behavior of a component, e.g., by using a strategy, eventually leads to an undesired effect: Assuming all links in the example shown in Fig. 1 are well-formed except one, the link between output port $O_{1,1}$ of component *FuelSensor* and input port $I_{2,1}$ of component *FuelController*. Using a strategy $\mathcal{A}'_{FC}$ for component *FuelController* instead of the original TIOA $\mathcal{A}_{FC}$ resolves this conflict. What should be prevented is being required to re-validate well-formed links from or to components being neighbored to component *FuelController*. For example, using a different TIOA for component *FuelController* can lead to the situation that the link between *FuelController* and *EngineModel* is no longer well-formed. This conflict can be solved by deriving a strategy for *EngineModel*, potentially invalidating another link connected to an output port of *EngineModel*. Such an effect can be propagated through the overall cyclic architecture resulting in changes that need to be applied multiple times on each component. Following it is shown that if strategies are used according to Def. 14, this effect cannot occur on output ports of the changed component.

By construction any resulting strategy $\mathcal{A}'$ of a TIOA $\mathcal{A}$ represents a subset of the reachable transitions of the associated timed-system. As a result also for the trace sets of each strategy $\mathcal{A}'$ holds $traces(\mathcal{A}') \subseteq traces(\mathcal{A})$. The same holds for the derived strategy for component *FuelController*: $traces(\mathcal{A}'_{FC}) \subseteq traces(\mathcal{A}_{FC})$. According to Def. 15 it can be concluded that using a strategy $\mathcal{A}'$ instead of $\mathcal{A}$ does not invalidate any well-formed link between an output port of a changed component and an input port of any other component. Consider the link between the output port $O_{2,1}$ of component *FuelController* and the input port $I_{3,1}$ of component *EngineModel*. If previously holds $traces(\mathcal{A}_{FC})\, pro\, \mathcal{A}_{EM}$ and for strategy $\mathcal{A}'_{FC}$ holds $traces(\mathcal{A}'_{FC}) \subseteq traces(\mathcal{A}_{FC})$, according to Def. 15 it follows that $traces(\mathcal{A}'_{FC})\, pro\, \mathcal{A}_{EM}$ and the link is still well-formed.

8.3 Evaluation Results - Complexity

In this section the results of the application example introduced in Sec. 3 are briefly discussed. Performance results in case of the explored states during the verification of the

overall architecture are investigated in more detail. The monolithic approach,[18] where all involved TIOA are analyzed at once, is compared with the modular approach, where only connected TIOA (linked components) are analyzed in pairs. In such a manner it is shown that the introduced reasoning scheme outperforms the monolithic approach by numbers and thus fulfills requirement (R1).

When checking deadlock freedom of the testbeds, UPPAAL provides the possibility to determine the number of symbolic states explored during verification. Table 1 shows the result in case of the states that have been explored for ensuring that the property of language progressive behavior is fulfilled for each involved TIOA. Using the approach proposed in this work (following called *modular* approach), the checking of each testbed requires at most 291 symbolic states and 564 symbolic states at all. A monolithic check, where all three TIOA are involved at once requires 184 symbolic states in UPPAAL. While for the application example the monolithic as well as the modular approach scales (verification can be applied in both cases within two seconds using UPPAAL), it is following shown that the modular approach outperforms the monolithic approach when using a higher number of components. A synthesized example has been used where a set of components are connected in a row. The number of states, when applying the monolithic and the modular approach, are compared.[19] The results are listed in Table 1, where the first column describes which example is used, the second column shows the number of included components, resp. TIOA, the third column shows the number of states explored using the monolithic approach and the last column shows the number of states when using the modular approach for checking deadlock freedom. Especially in case of the synthesized example, the exponential growth in case of the explored states becomes visible. The check including 15 components requires more than half an hour to be verified using the monolithic approach.[20] Any example using 18 components or more consumes more working memory than provided by the used PC when applying the monolithic check. In contrast, the modular approach requires less than 1 second for each check.

Table 1: Verification type and resulting number of states.

Example	**#Components**	**#States Monolithic**	**#States Modular**
FuelControl	3	184	249 + 24 + 291=564
Synthesized	6	1.608	1.128
Synthesized	9	16.734	1.692
Synthesized	12	222.236	2.256
Synthesized	15	3.683.147	2.820
Synthesized	18	out of memory	3.384

Figure 28 graphically shows the difference between the modular and the monolithic approach. While the modular approach rarely differs concerning the required states that need to be explored for ensuring well-formedness, the monolithic approach clearly shows an exponential growth, depending on the number of used components. Especially in the case that

[18] The approach by construction does not fulfill requirement R3 (protecting IPs).

[19] The synthesized example consists of an architecture where multiple instances of the previously shown components of the fuel rate control system are used in form of a larger architecture of pairwaise linked components.

[20] All checks have been executed using the 64 bit version of UPPAAL running on a 2.4 GHz dual-core PC with 8 GB of RAM.

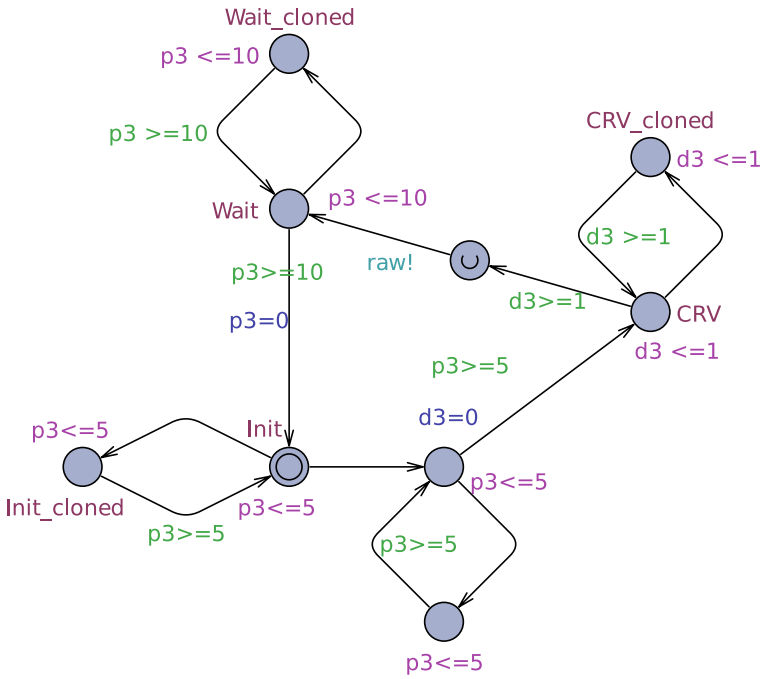


Figure 27: TIOA $\mathcal{A}^c_{EM}$ for testing language progressiveness of TIOA $\mathcal{A}_{FS}$ representing component *FuelSensor* like shown in Fig. 5.

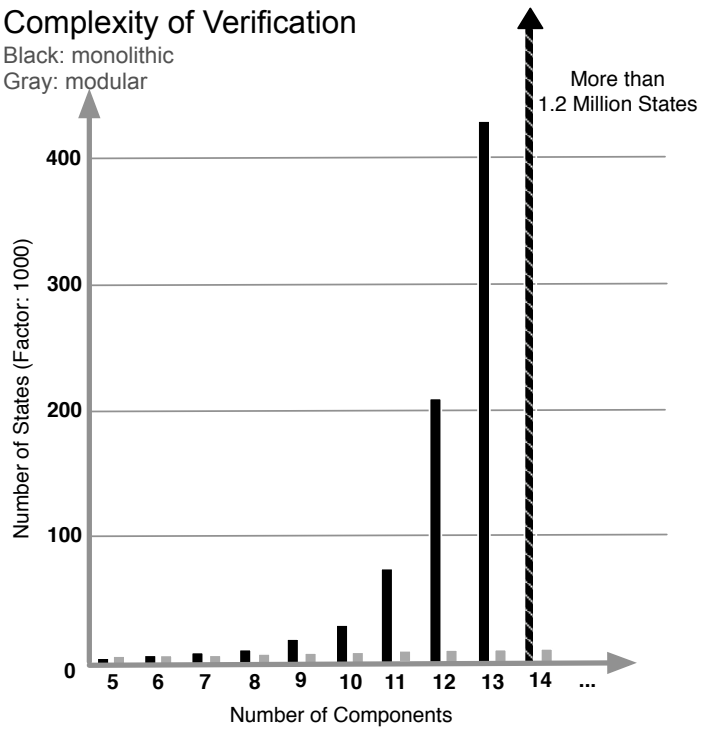


Figure 28: Explored states during analysis using the monolithic and the modular approach.

more than 11 components are included in the architecture, the monolithic approach requires more than 200000 states to be explored at once. In the case of 14 components more than 1.2 million states are explored. Thus, all architectures including more than 14 components are skipped in Fig. 28 because the verification requires more states to be explored than can be shown in the scale used in Fig. 28.

9 Abstraction

In the following, it is briefly discussed how abstraction techniques can be used in our reasoning scheme for protecting IPs, supporting requirement (R3).

9.1 Automatic Abstraction for IP Protection

One requirement for several real-time embedded systems is (R3): support for protecting IPs such that internal details of individual components do not need to be disclosed during well-formedness checks. Consider the well-formedness check of the component *FuelController* using $\mathcal{A}_{FS}$. $\mathcal{A}_{FS}$ is the behavior model of the linked component *FuelSensor*. It is used as the representation for the languages of the input ports $I_{2,1}$ and $I_{2,2}$ of the component *FuelController* shown in Fig. 26. In the case *FuelSensor* and *FuelController* are developed by two different suppliers, the supplier developing the first component is not willing to provide internal details related to Intellectual Properties (IP) to the second supplier. The second supplier develops the component *FuelController* and relies on $\mathcal{A}_{FS}$ or any other valid representation of the input languages for being able to apply the well-formedness check. In the following, it is shown how such a conflict, that the second supplier of component *FuelController* relies on the behavior model including IPs of the first supplier, can be resolved.

It is assumed that IPs are mainly related to internal signals and by removing these signals from $\mathcal{A}_{FS}$, like done when using $\mathcal{A}_{FS}$ for creating a testbed, these properties are well protected in our framework. Nevertheless, potentially IPs are associated with other syntactical elements of the component *FuelSensor*, represented by TIOA $\mathcal{A}_{FS}$. Examples of such elements are locations, edges or clock variables. Protecting IPs means hiding these syntactical elements by creating an abstract representation for the component *FuelSensor*, allowing to be used during well-formedness checks without providing syntactical elements related to IPs. Such an abstract representation of component *FuelSensor* is following called $\mathcal{A}_{FS/}$. If an abstract representation $\mathcal{A}_{FS/}$ of $\mathcal{A}_{FS}$ can be derived for that holds $traces(\mathcal{A}_{FS}) \subseteq traces(\mathcal{A}_{FS/})$, according to Def. 15, the abstract version $\mathcal{A}_{FS/}$ can be taken as a representation of the input language for component *FuelController*.

Thus, if the well-formedness constraints are fulfilled for the abstract representation $\mathcal{A}_{FS/}$ they are also fulfilled for the more detailed version $\mathcal{A}_{FS}$. As a result of Def. 15 holds, if $\mathcal{A}_{FS/} \restriction I_{3,1} \, pro \, \mathcal{A}_{FC}$, with $traces(\mathcal{A}_{FS}) \subseteq traces(\mathcal{A}_{FS/})$ implies $\mathcal{A}_{FS} \restriction I_{3,1} \, pro \, \mathcal{A}_{FC}$. Thus, any abstraction mechanism that can be applied on a TIOA resulting in $traces(\mathcal{A}_{FS}) \subseteq traces(\mathcal{A}_{FS/})$ is permitted. An example of an existing technique allowing deriving such an

abstract representation $\mathcal{A}_{FS/}$ based on $\mathcal{A}_{FS}$ is given in [21]. In [21] the authors show how to derive an abstract representation of a TA A by removing syntactical elements. According to [21] the shown abstraction mechanism guarantees that for any abstract version $A_{/}$ of any given TA A holds: $traces(A) \subseteq traces(A_{/})$. Thus, our framework and reasoning scheme can be adapted by existing techniques for further support requirement (R3) if desired. How such techniques can be further integrated into the framework described in this work, e.g., for resolving conflicts in case errors are introduced by the abstraction, is considered to be future work.

10 Conclusion and Future Work

The concept of language progressive and language receptive timed I/O automata has been introduced as a compromise between the existing purely optimistic and completely pessimistic approaches.

If we consider as input language all zeno-free and deadlock-free traces over the input signals of a given timed I/O automaton, our notion of language progressiveness and language receptiveness coincide with the pessimistic approach. Thus, the pessimistic approach is a special case of our more general framework for timed I/O automata.

We further demonstrated how TIOA can be used to set-up a compositional reasoning scheme that supports component-based architectures of embedded real-time systems. This reasoning scheme removes the limitations of both extreme perspectives and therefore satisfies requirement (R1), (R2), (R3) and (R4) as stated in Section 1. Specifically, our compromise between the too pessimistic approaches that avoid any reliance on the characteristics of the environment and the too optimistic approaches, which too much rely on helpful environment characteristics, permits us finding a solution that scales (R1) but avoids the need for too strong assumptions on the available resources (R2). By using an additional abstraction step, also the IP of component developers is protected (R3). Further the introduced approach is expressive enough to cover arbitrary structures, including cyclic dependencies (R4). As future work, we plan developing tool support for the approach and integrate it with available architectural frameworks such as AUTOSAR. As a first step we have shown in [19] how to automatically derive TA models from given AUTOSAR architecture descriptions.

References

[1] Luca Alfaro and Thomas A. Henzinger. Interface Automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, 2001.

[2] Luca Alfaro and Thomas A. Henzinger. Interface Theories for Component-Based Design. In *Proc. of 1st IWoES*, volume 2211 of *LNCS*. Springer, 2001.

[3] Luca Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Timed Interfaces. In *Proc. of EMSOFT '02*. Springer, 2002.

[4] Rajeev Alur and David L. Dill. A Theory of Timed Automata. In *extended version of the ICALP90 paper*, 1990.

[5] Eugene Asarin, Oded Maler, and A. Pnueli. Symbolic Controller Synthesis for Discrete and Timed Systems. In Panos Antsaklis, W. Kohn, A. Nerode, and Shankar Sastry, editors, *Hybrid System II*, LNCS. Springer, 1995.

[6] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-Time Components in BIP. In *Proc. SEFM'06*. IEEE Computer Society, 2006.

[7] Gerd Behrmann, Agnes Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. UPPAAL-Tiga: Time for Playing Games! In Werner Damm and Holger Hermanns, editors, *In Proc. of CAV'07*, volume 4590 of *LNCS*. Springer, July 2007.

[8] Gerd Behrmann, Alexandre David, and Kim Larsen. A Tutorial on Uppaal. In *Proc. of SFM-04:RT*, LNCS. Springer, 2004.

[9] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In *CONCUR 2005 Concurrency Theory*, volume 3653 of *LNCS*. Springer, 2005.

[10] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed I/O Automata: A Complete Specification Theory for Real-Time Systems. In *Proc. of HSCC '10*. ACM, 2010.

[11] Henrik Ejersbo Jensen, Kim Guldstrand Larsen, and Arne Skou. Scaling up Uppaal: Automatic Verification of Real-Time Systems using Compositionality and Abstraction. In *Proc. of FTRTFT*, volume 1926 of *LNCS*. Springer, 2000.

[12] Holger Giese, Stefan Neumann, Oliver Niggemann, and Bernhard Schätz. Model-based integration. In *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems*, MBEERTS'07. Springer, 2010.

[13] Susanne Graf and Sophie Quinton. Contracts for BIP: Hierarchical Interaction Models for Compositional Verification. In *Proc. of FORTE*. Springer, 2007.

[14] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Inf. Comput.*, 111(2):193–244, 1994.

[15] Bengt Jonsson, Simon Perathoner, Lothar Thiele, and Wang Yi. Cyclic Dependencies in Modular Performance Analysis. In *Proc. EMSOFT '08*. ACM, 2008.

[16] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata (Synthesis Lectures in Computer Science)*. Morgan & Claypool, 2006.

[17] Kai Lampka, Simon Perathoner, and Lothar Thiele. Analytic Real-Time Analysis and Timed Automata: A Hybrid Method for Analyzing Embedded Real-Time Systems. In *Proc. of EMSOFT '09*. ACM, 2009.

[18] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symposium*. IEEE Computer Society, 1995.

[19] Stefan Neumann, Norman Kluge, and Sebastian Wätzoldt. Automatic Transformation of Abstract AUTOSAR Architectures to Timed Automata. In *Proc. of the 5th ACES-MB'12 Workshop*. ACM, 2012.

[20] Sophie Quinton and Susanne Graf. Contract-based verification of hierarchical systems of components. In *Proc. of SEFM'08*. IEEE Computer Society, 2008.

[21] Ramzi Ben Salah, Marius Bozga, and Oded Maler. On Timed Components and Their Abstraction. In *Proc. of SIGSOFT'07*. ACM, 2007.

[22] Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. Real-Time Interfaces for Composing Real-Time Systems. In *Proc. of EMSOFT '06*. ACM, 2006.

A Proofs

First we give a definition what is following called a *cyclic blocking language progressive* pair of two TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$.

Definition 30
Given two compatible and valid TIOA $\mathcal{A}_1 = (A_1, I_1, O_1, H_1)$, $\mathcal{A}_2 = (A_2, I_2, O_2, H_2)$ with $O_1 = I_2$, $O_2 = I_1$, $\mathcal{A}_1 \restriction O_1 \, pro \, \mathcal{A}_2$ and $\mathcal{A}_2 \restriction O_2 \, pro \, \mathcal{A}_1$. Let $\mathcal{T}_{1,2} = (\Sigma_{1,2}, \mathcal{S}_{1,2}, \mathcal{S}^0_{1,2}, T_{1,2})$ be the induced timed system of $\mathcal{A}_1 \parallel \mathcal{A}_2$, $\mathcal{T}_{1,-} = (\Sigma_{1,-}, \mathcal{S}_{1,-}, \mathcal{S}^0_{1,-}, T_{1,-})$ be the induced timed system of $\mathcal{A}_1 \restriction O_1 \parallel \mathcal{A}_2$ and $\mathcal{T}_{-,2} = (\Sigma_{-,2}, \mathcal{S}_{-,2}, \mathcal{S}^0_{-,2}, T_{-,2})$ the induced timed system of $\mathcal{A}_1 \parallel \mathcal{A}_2 \restriction O_2$. Let $\mathcal{T}_1 = (\Sigma_1, \mathcal{S}_1, \mathcal{S}^0_1, T_1)$ be the induced timed system of $\mathcal{A}_1$ and $\mathcal{T}_2 = (\Sigma_2, \mathcal{S}_2, \mathcal{S}^0_2, T_2)$ the induced timed system of $\mathcal{A}_2$. We call $\mathcal{A}_1$ and $\mathcal{A}_2$ cyclic blocking language progressive*, if $\mathcal{S}_{1,2}$ includes a deadlock state s_d.*

For such a *cyclic blocking language progressive pair* $\mathcal{A}_1$ and $\mathcal{A}_2$ each deadlock state s_d is also reachable in the parallel product of $\mathcal{A}_1 \restriction O_1 \parallel \mathcal{A}_2$.

Lemma A.1
Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two cyclic blocking language progressive TIOA according to Def. 30. For each deadlock state $s_d \in \mathcal{S}$ holds: $s_d \in \mathcal{S}_{1,-}$ and $s_d \in \mathcal{S}_{-,2}$.

Proof A.1
(sketch) The only differences between $\mathcal{A}_1 \parallel \mathcal{A}_2$ and $\mathcal{A}_1 \parallel \mathcal{A}_2 \restriction O_2$, resp. $\mathcal{A}_1 \restriction O_1 \parallel \mathcal{A}_2$ is, that signals are removed in the later. By construction signals can only prevent transitions in a parallel product but not enable (cf. Def. 3) a single transition. When removing signals, transitions can only be added in $\mathcal{A}_1 \parallel \mathcal{A}_2 \restriction O_2$, resp. $\mathcal{A}_1 \restriction O_1 \parallel \mathcal{A}_2$ and as a consequence all states reachable in $\mathcal{A}_1 \parallel \mathcal{A}_2$ are also reachable in $\mathcal{A}_1 \parallel \mathcal{A}_2 \restriction O_2$, resp. $\mathcal{A}_1 \restriction O_1 \parallel \mathcal{A}_2$.

Further, for each deadlock state s_d no transition including the empty signal can be contained neither in the state s_1 of $\mathcal{A}_1$ when being in the deadlock, nor in the state s_2 of $\mathcal{A}_2$ when being the same deadlock state s_d.

Lemma A.2
Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two cyclic blocking language progressive TIOA according to Def. 30. Let s_d be a deadlock state. s_d is of form $s_d = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$, with l_1 being a location of A_1, l_2 being a location of A_2, v_1 a variable assignment for all variables of A_1 and v_2 a variable assignment for all variables of A_2. When being in state s_d of $\mathcal{S}_{1,2}$, $\mathcal{A}_1$ is in state $s_1 = \langle l_1, v_1\rangle$ and $\mathcal{A}_2$ is in state $s_2 = \langle l_2, v_2\rangle$. Let $T_1(s_1)$ be the set of transitions in state s_1 and $T_2(s_2)$ be the set of transitions in state s_2. It holds:

$\neg\exists t \in T_1(s_1) \cup T_2(s_2)$ *with* $t = s_i \xrightarrow{\sigma} s'_i$, $i \in \{1, 2\}$ *and* $\sigma = \sigma_\epsilon$.

Proof A.2
Assuming $\exists t \in T_1(s_1) \cup T_2(s_2)$ with $\sigma_i = \sigma_\epsilon$. Because t is a discrete transition that do not need to be synchronized with any signal it needs to be preserved in $s_d = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$ and $T_{1,2}(s_d)$ need to include this transition. This is the case because $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible

TIOA that do not share any clocks, attributes or constraints over the same clocks or attributes. As a consequence s_d cannot be a deadlock state and we have a contradiction.

Like following shown in Lemma A.3, transitions are included in those transitions observable at state s_1 of $\mathcal{A}_1$ and in s_2 of $\mathcal{A}_2$ (when being in a deadlock state s_d), such that signals from the input as well as the output of both TIOA can be observed.

Lemma A.3
Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two cyclic blocking language progressive TIOA according to Def. 30. Let s_d be a deadlock state. s_d is of form $s_d = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$, with l_1 being a location of A_1, l_2 being a location of A_2, v_1 a variable assignment for all variables of A_1 and v_2 a variable assignment for all variables of A_2. When being in state s_d of $\mathcal{S}_{1,2}$, $\mathcal{A}_1$ is in state $s_1 = \langle l_1, v_1\rangle$ and $\mathcal{A}_2$ is in state $s_2 = \langle l_2, v_2\rangle$. Let $T_1(s_1)$ be the set of transitions in state s_1 and $T_2(s_2)$ be the set of transitions in state s_2. It holds:

$\exists t_{O_1}, t_{O_2} \in T_1(s_1) \cup T_2(s_2)$ *with* $t_{O_1} = s_{O_1} \xrightarrow{\sigma_1} s'_{O_1}$, $t_{O_2} = s_{O_2} \xrightarrow{\sigma_2} s'_{O_2}$, $\sigma_1 \in O_1$ *and* $\sigma_2 \in O_2$.

Proof A.3
Because $\mathcal{A}_1{\restriction}O_1 \, pro \, \mathcal{A}_2$ and $\mathcal{A}_2{\restriction}O_2 \, pro \, \mathcal{A}_1$, it holds that $\mathcal{A}_1{\restriction}O_1 \parallel \mathcal{A}_2$ and $\mathcal{A}_2{\restriction}O_2 \parallel \mathcal{A}_1$ are deadlock free. Because of Lemma A.1 it holds that each deadlock state s_d observable on $\mathcal{A}_1 \parallel \mathcal{A}_2$ is also observable on $\mathcal{A}_1{\restriction}O_1 \parallel \mathcal{A}_2$ and $\mathcal{A}_2{\restriction}O_2 \parallel \mathcal{A}_1$, where s_d is no longer a deadlock state. Because in $\mathcal{A}_1{\restriction}O_1 \parallel \mathcal{A}_2$ only signals from O_2 are removed and s_d is no longer a deadlock state, at least one transition is enabled in s_d by removing a signal included in O_2. Thus, a transition $t_{O_2} = s_{O_2} \xrightarrow{\sigma_2} s'_{O_2}$ need to be included in $T_1(s_1)$ or $T_2(s_2)$. The same holds for the case $\mathcal{A}_2{\restriction}O_2 \parallel \mathcal{A}_1$ where only signals included in O_1 are removed and as a consequence also a transition $t_{O_1} = s_{O_1} \xrightarrow{\sigma_1} s'_{O_1}$ with $\sigma_1 \in O_1$ need to be included in $T_1(s_1)$ or $T_2(s_2)$.

In Lemma A.4 it is following shown that in the case a deadlock state s_d is reachable in the induced timed system of $\mathcal{A}_1 \parallel \mathcal{A}_2$, for the two valid and compatible TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$ it cannot hold that both are able taking a continuous transition in $s_1 = \langle l_1, v_1\rangle$ and $s_2 = \langle l_2, v_2\rangle$ of $s_d = \langle\langle l_1, v_1\rangle, \langle l_2, v_2\rangle\rangle$.

Lemma A.4
Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two cyclic blocking language progressive TIOA according to Def. 30. Let s_d be a deadlock state. s_d is of form $s_d = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$, with l_1 being a location of A_1, l_2 being a location of A_2, v_1 a variable assignment for all variables of A_1 and v_2 a variable assignment for all variables of A_2. When being in state s_d of $\mathcal{S}_{1,2}$, $\mathcal{A}_1$ is in state $s_1 = \langle l_1, v_1\rangle$ and $\mathcal{A}_2$ is in state $s_2 = \langle l_2, v_2\rangle$. Let $T_1(s_1)$ be the set of transitions in state s_1 and $T_2(s_2)$ be the set of transitions in state s_2. It holds:

$\neg\exists t_1, t_2$, *with* $t_1 \in T_1(s_1)$, $t_1 = s_1 \xrightarrow{\tau} s'_1$, $t_2 \in T_2(s_2)$, $t_2 = s_2 \xrightarrow{\tau} s'_2$ *and* $\tau \in \mathbb{R}^+$ *being two continuous transitions.*

Proof A.4
Assuming t_1 and t_2 exist being continuous transitions of s_1 resp. s_2. $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible TIOA that do not share any clock variable, attribute variable or any constraints about common variables. As a consequence sets of invariants and guards are also disjoint for

the individual TIOA and in state s_d of the induced timed system of $\mathcal{A}_1 \parallel \mathcal{A}_2$ a continuous transition need to exist. Thus, s_d cannot be a deadlock state and we have a contradiction.

The previously shown lemmata are used in Lemma A.5 showing that Theorem 6.4 (see Sec. 6.2) holds.

Lemma A.5
Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two cyclic blocking language progressive TIOA according to Def. 30. Let s_d be a deadlock state. s_d is of form $s_d = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$, with l_1 being a location of A_1, l_2 being a location of A_2, v_1 a variable assignment for all variables of A_1 and v_2 a variable assignment for all variables of A_2. When being in state s_d of $\mathcal{S}_{1,2}$, $\mathcal{A}_1$ is in state $s_1 = \langle l_1, v_1\rangle$ and $\mathcal{A}_2$ is in state $s_2 = \langle l_2, v_2\rangle$. Let $T_1(s_1)$ be the set of transitions in state s_1 and $T_2(s_2)$ be the set of transitions in state s_2. It holds:

For $\mathcal{A}_i$ with $i \in \{1, 2\}$: $\exists s_i, \alpha_i$ with $s_i \in \mathcal{S}_i$, $\alpha_i \in traces(\mathcal{A}_i)$, $\alpha_i = \beta_i \circ \gamma_i \circ \delta_i$, $\beta_i \in traces\_to(s_i, \mathcal{A}_i)$, $time(\gamma_i) = 0$, $signals(\gamma_i) \cap I_i \neq \emptyset$ and $signals(\gamma_i) \cap O_i \neq \emptyset$.

Proof A.5
First, the property is shown for $\mathcal{A}_i = \mathcal{A}_2$: For $\mathcal{A}_2$ we take $s_2 = \langle l_2, v_2\rangle$ as state s_i for that holds an $\alpha_2 \in traces(A_2)$ exists with $\alpha_2 = \beta_2 \circ \gamma_2 \circ \delta_2$ and $\beta_2 \in traces\_to(s_2, \mathcal{A}_2)$. This is the case because the deadlock state $s_d = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$ is reachable in $\mathcal{A}_1 \parallel \mathcal{A}_2$ and due to the fact that synchronization only removes reachable states in our model of TIOA, s_2 need to be reachable via a trace $\beta_2 \in traces\_to(s_2, \mathcal{A}_2)$. Because $\mathcal{A}_2$ is deadlock free, a trace μ need to be included in $traces\_from(s_2, \mathcal{A}_2)$. We can split this trace into two parts $\mu = \gamma_2 \circ \delta_2$ resulting in $\alpha_2 = \beta_2 \circ \gamma_2 \circ \delta_2$. Based on Lemma A.3 we can distinguish three different cases for $\mathcal{A}_1$, the other TIOA, when being in state s_d:

- *1) $\exists t_1 \in T_1(s_1)$ with $t_1 = s_1 \xrightarrow{\sigma_1} s'_a$ for that holds $\sigma_1 \in O_1$ and $\neg\exists t_2 \in T_1(s_1)$ with $t_2 = s_1 \xrightarrow{\sigma_2} s'_b$ for that holds $\sigma_2 \in O_2$,*
- *2) $\neg\exists t_1 \in T_1(s_1)$ with $t_1 = s_1 \xrightarrow{\sigma_1} s'_a$ for that holds $\sigma_1 \in O_1$ and $\exists t_2 \in T_1(s_1)$ with $t_2 = s_1 \xrightarrow{\sigma_2} s'_b$ for that holds $\sigma_2 \in O_2$ or*
- *3) $\exists t_1 \in T_1(s_1)$ with $t_1 = s_1 \xrightarrow{\sigma_1} s'_a$ for that holds $\sigma_1 \in O_1$ and $\exists t_2 \in T_1(s_1)$ with $t_2 = s_1 \xrightarrow{\sigma_2} s'_b$ for that holds $\sigma_2 \in O_2$*

The case 4): $\neg\exists t_1 \in T_1(s_1)$ with $t_1 = s_1 \xrightarrow{\sigma_1} s'_a$ for that holds $\sigma_1 \in O_1$ and $\neg\exists t_2 \in T_1(s_1)$ with $t_2 = s_1 \xrightarrow{\sigma_2} s'_b$ for that holds $\sigma_2 \in O_2$ is not possible because of Lemma A.3. Following we show for all three possible cases of $\mathcal{A}_1$ individually that each individual case implicates that Lemma A.5 need to be fulfilled for $\mathcal{A}_2$ if a deadlocks state s_d exists:

1) Assuming the case: $\exists t_1 \in T_1(s_1)$ with $t_1 = s_1 \xrightarrow{\sigma_1} s'_a$ for that holds $\sigma_1 \in O_1$ and $\neg\exists t_2 \in T_1(s_1)$ with $t_2 = s_1 \xrightarrow{\sigma_2} s'_b$ for that holds $\sigma_2 \in O_2$. Because of Lemma A.1 it follows that state $s_d = \langle\langle l_1, l_2\rangle, \langle v_1, v_2\rangle\rangle$ is reachable in $\mathcal{A}_1{\restriction}O_1 \parallel \mathcal{A}_2$, where $\mathcal{A}_1$ is in state $s_1 = \langle l_1, v_1\rangle$ and $\mathcal{A}_2$ is in state $s_2 = \langle l_2, v_2\rangle$. Because $T_1(s_1)$ does not contain a transition $t_2 = s_1 \xrightarrow{\sigma_2} s'_b$ with $\sigma_2 \in O_2$, according to Lemma A.3 it follows $\exists t \in T_2(s_2)$ with $t = s_2 \xrightarrow{\sigma_2} s'_2$ and $\sigma_2 \in O_2$.

Thus, a trace $\alpha_2 = \beta_2 \circ \sigma_2 \circ \delta_2$ exists with $\beta_2 \in traces\_to(s_2, \mathcal{A}_2)$ where $\mathcal{A}_2$ allows observing $\sigma_2 \in O_2$. Because $\mathcal{A}_1 \uparrow O_1 \, pro \, \mathcal{A}_2$, s_d is no deadlock state in $\mathcal{A}_1 \uparrow O_1 \parallel \mathcal{A}_2$, where $\mathcal{A}_2$ is in state s_2 and $\mathcal{A}_1$ is in state s_1. Because of the language progressiveness ($\mathcal{A}_1 \uparrow O_1 \, pro \, \mathcal{A}_2$, cf. Def. 15 and Theorem 5.1) $\mathcal{A}_1$ is able to take the discrete transition at state s_1 including signal σ_1 not allowing time to diverge till $\mathcal{A}_2$ has received this signal. Because s_d is a deadlock in $\mathcal{A}_1 \parallel \mathcal{A}_2$, σ_1 cannot be received directly in state s_2 (otherwise s_d want be a deadlock state). As a consequence a successor state $s_2^{'n}$ of s_2 need to exist, reachable in zero time allowing observing a trace κ with $time(\kappa) = 0$, where σ_1 is included in transition $T_2(s_2^{'n})$. Thus, s_2 is an example of $\mathcal{A}_2$ with $s_2 \in \mathcal{S}_2$, $\exists \alpha_2 \in traces(\mathcal{A}_2)$, $\alpha_2 = \beta_2 \circ \sigma_2 \circ \kappa \circ \sigma_1$, $\beta_2 \in traces\_to(s_2, \mathcal{A}_2)$ and $time(\kappa) = 0$. By choosing $\sigma_2 \circ \kappa \circ \sigma_1 = \gamma_2$ the property of Lemma A.5 is fulfilled for this case.

2) Assuming $\neg \exists t_1 \in T_1(s_1)$ with $t_1 = s_1 \xrightarrow{\sigma_1} s'_a$ for that holds $\sigma_1 \in O_1$ and $\exists t_2 \in T_1(s_1)$ with $t_2 = s_1 \xrightarrow{\sigma_2} s'_b$ for that holds $\sigma_2 \in O_2$. Because of Lemma A.4 we can subdivide this case into two sub-cases: 2 a) $T_1(s_1)$ includes no continuous transition and 2 b) $T_1(s_1)$ contains a continuous transition.

Assuming 2 a): Because of Lemma A.2 the set of transitions $T_1(s_1)$ only contains discrete transitions including signals of O_2. Because of Lemma A.5 follows $T_2(s_2)$ need to contain a discrete transition including a signal $\sigma_1 \in O_1$. Because of $\mathcal{A}_2 \uparrow O_2 \, pro \, \mathcal{A}_1$ and s_1 being reachable in $\mathcal{A}_2 \uparrow O_2 \parallel \mathcal{A}_1$ (s_d is reachable in $\mathcal{A}_2 \uparrow O_2 \parallel \mathcal{A}_1$), s_1 is not allowed to lead to a deadlock in $\mathcal{A}_2 \uparrow O_2 \parallel \mathcal{A}_1$. For $\mathcal{A}_2 \uparrow O_2 \parallel \mathcal{A}_1$ being in state s_b, where $\mathcal{A}_1$ is in state s_1, it is not possible to let diverge time (because in the considered case no continuous transition is containing in $T_1(s_1)$) but only to receive signals from O_2. Thus, there need to exist a successor state $s_2^{'n}$ of s_2 being reachable in zero time via a trace κ with $time(\kappa) = 0$ for that holds, $T_2(s_2^{'n})$ includes a transition sending a signal $\sigma \in O_2$. Again, we can construct for $s_2 \in \mathcal{S}_2$ a trace α_2 with $\alpha_2 \in traces(\mathcal{A}_2)$, with $\alpha_2 = \beta_2 \circ \sigma_1 \circ \kappa \circ \sigma$ and $\beta_2 \in traces\_to(s_2, \mathcal{A}_2)$. By renaming $\sigma_1 \circ \kappa \circ \sigma = \gamma_2$, s_2 contains a zero-time input/output trace.

Assuming 2 b): Because of Lemma A.4, $T_2(s_2)$ cannot contain a continuous transition. Because of $\mathcal{A}_1 \uparrow O_1 \, pro \, \mathcal{A}_2$ no deadlock can exist in $\mathcal{A}_1 \uparrow O_1 \parallel \mathcal{A}_2$ also in the case $\mathcal{A}_1$ in state s_1 takes the continuous transition while $\mathcal{A}_2$ is in state s_2 where no continuous transition can be taken. Thus, in s_2 a discrete transition need to be included that not contains a signal of O_1 ($T_1(s_1)$ contains no such transition in s_d while synchronizing all signals of O_1) and that not contains the empty signal (cf. Lemma A.2). As a consequence at least one transition of $T_2(s_2)$ need to contain one or more discrete transitions including a signal of O_2 while no transition of $T_1(s_1)$ contains this signal (otherwise s_d won't be a deadlock in $\mathcal{A}_1 \parallel \mathcal{A}_2$). Thus, s_d is a state where $\mathcal{A}_2$ is in state s_2 allowing observing one or more signals included in O_2, while $\mathcal{A}_1$ is in state s_1 not allowing receiving any of these signal and not allowing taking another discrete transition (only transitions including signals from O_1 are observable on s_1 according to the considered case). Because $\mathcal{A}_2$ in state s_2 allows to observe a signal included in O_2 that cannot be received by any transition in s_1 of $\mathcal{A}_1$, and in s_1 only transitions can be included that need to synchronize with signals of O_2, the property $\mathcal{A}_2 \uparrow O_2 \, pro \, \mathcal{A}_1$ cannot be fulfilled and we have a contradiction. As a consequence this case cannot occur for two cyclic blocking language progressive TIOA and do not need to be considered here.

3) Assuming $\exists t_1 \in T_1(s_1)$ with $t_1 = s_1 \xrightarrow{\sigma_1} s'_a$ for that holds $\sigma_1 \in O_1$ and $\exists t_2 \in T_1(s_1)$ with

$t_2 = s_1 \xrightarrow{\sigma_2} s'_b$ for that holds $\sigma_2 \in O_2$. In state s_d (a deadlock state of $\mathcal{A}_1 \parallel \mathcal{A}_2$), let $\mathcal{A}_1$ be in state s_1 and $\mathcal{A}_2$ be in state s_2. A transition need to be included in $T_1(s_1)$ containing a signal $\sigma_1 \in O_1$. Because s_d is a deadlock state in $\mathcal{A}_1 \parallel \mathcal{A}_2$, $T_2(s_2)$ cannot contain a transition with the same signal. Because $\mathcal{A}_1 \upharpoonright O_1 \, pro \, \mathcal{A}_2$, $\mathcal{A}_1$ need to be able to send σ_1 when being in state s_1 without taking any other transition (in any other case $\mathcal{A}_2$ contains a transition blocking state and $\mathcal{A}_1 \upharpoonright O_1 \, pro \, \mathcal{A}_2$ cannot be fulfilled). Because in $T_2(s_2)$ this signal cannot be consumed, a transition need to be taken in $T_2(s_2)$ including a signal $\sigma_3 \in O_2$ ($T_2(s_2)$ cannot contain a transition including the signal σ_1, because otherwise s_d want be a deadlock state in $\mathcal{A}_1 \parallel \mathcal{A}_2$). It can also not be another transition including a signal of O_1 because all signals included in O_1 are synchronized and $\mathcal{A}_1$ can decide to only send σ_1 without causing a deadlock (because of the language progressiveness). Further, because of Lemma A.2 it cannot be a transition including the empty signal. Thus, in state s_2 a transition need to be observable that contains a $\sigma_3 \in O_2$, leading to s'_2. Further, in state s'_2 either a transition need to exist including signal σ_1 or a successor state s'^n_2 need to be reachable, without the passage of time, where σ_1 can be consumed (in any other case $\mathcal{A}_1 \upharpoonright O_1 \, pro \, \mathcal{A}_2$ cannot be fulfilled). As a consequence in state s_2 a signal $\sigma_3 \in O_2$ can be send and a successor state s'^n_2 of s_2 need to be reachable that allows to receive σ_1 while time do not need to progress between s_2 and s'^n_2. As a consequence a trace κ with $time(\kappa) = 0$ is observable between s_2 and s'^n_2. By choosing $\alpha_2 = \beta_2 \circ \sigma_3 \circ \kappa \circ \sigma_1$ with $\beta_2 \in traces\_to(s_2, \mathcal{A}_2)$ and $\gamma_2 = \sigma_3 \circ \kappa \circ \sigma_1$ it follows that Lemma A.5 is fulfilled for that last case.

Till now it has been shown that for each and every possible situation of $\mathcal{A}_1$, when being in a deadlock state s_d of the two cyclic blocking language progressive TIOA $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathcal{A}_2$ need to be in a state where the property of Lemma A.5 is fulfilled. By exchanging in the previously shown proof $\mathcal{A}_1$ with $\mathcal{A}_2$, it follows that for each and every possible case of $\mathcal{A}_2$, when being in a deadlock state s_d, also $\mathcal{A}_1$ is in a state according to Lemma A.5. As a consequence in each and every case a deadlock is included in the parallel product of two cyclic connected language progressive TIOA, it holds that for each involved TIOA the property is fulfilled and thus Lemma A.5 is fulfilled in general.

B Detailed TIOA models

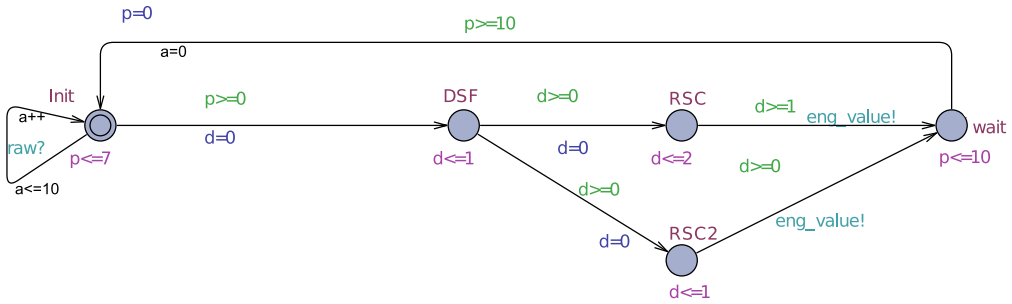


Figure 29: TIOA representing the component *FuelSensor* without providing undesired zeno behavior.

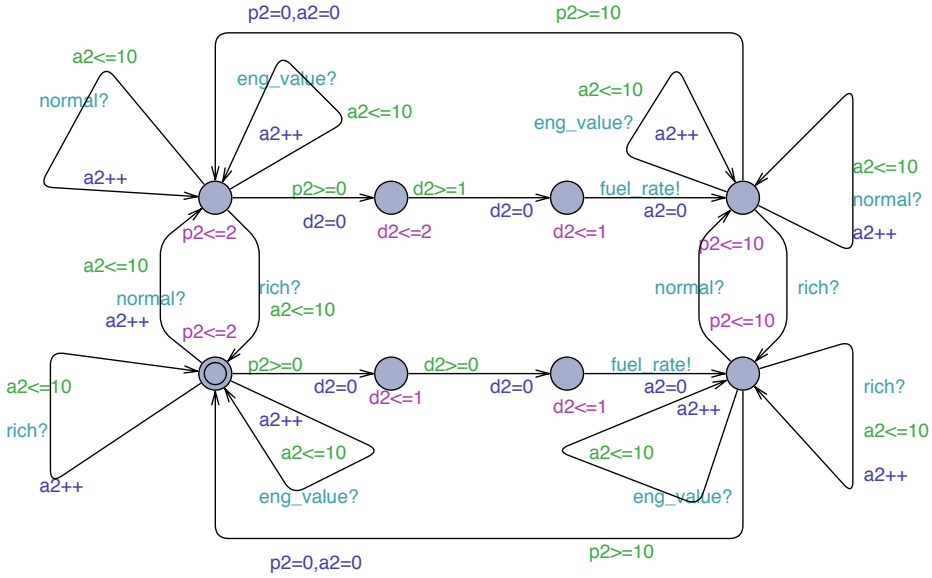


Figure 30: TIOA representing the component *FuelController* without providing undesired zeno behavior.

C Detailed Models of Testbeds

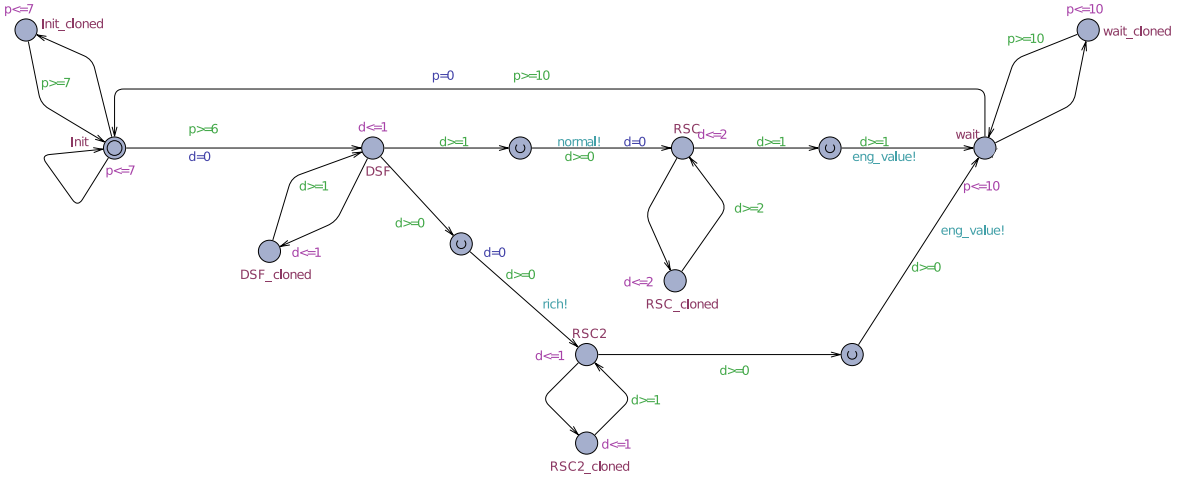


Figure 31: TIOA $\mathcal{A}^c_{FS}$ for checking if $\mathcal{A}^c_{FS}\, pro\, \mathcal{A}_{FC}$.

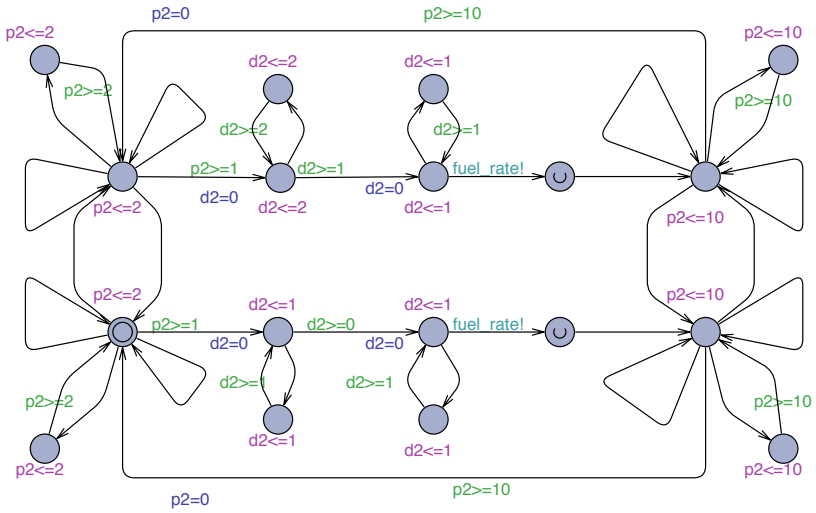


Figure 32: TIOA representing the first part of the testbed $TB_{EM}(\mathcal{A}_{FC}, \mathcal{A}_{EM})$.

D Combined Ports

Consider a given component for that holds two input ports are linked to output ports of a single component, like shown in Fig. 1 in case of connected ports of component *FuelSensor* and *FuelController*. The two input ports of component *FuelController* can be understood as a single input port. The same holds for the output ports of component *FuelSensor* that can be understood as a single combined port at the syntactical level by defining port $O_{1,3}$ representing the combined ports $O_{1,1}$ and $O_{1,2}$. For the signal set of the combined port holds $O_{1,3} = O_{1,1} \cup O_{1,2}$. The same can be done for the two input ports $I_{2,1}$ and $I_{2,2}$ by creating the syntactically combined port $I_{2,3} = I_{2,1} \cup I_{2,2}$. Accordingly only one link remains in the architecture and as a consequence both ports can be checked using only a single analysis step by only applying syntactical modifications on the components. The syntactically transformed architecture of the Fuel-Rate-Control architecture is shown in Fig. 33.

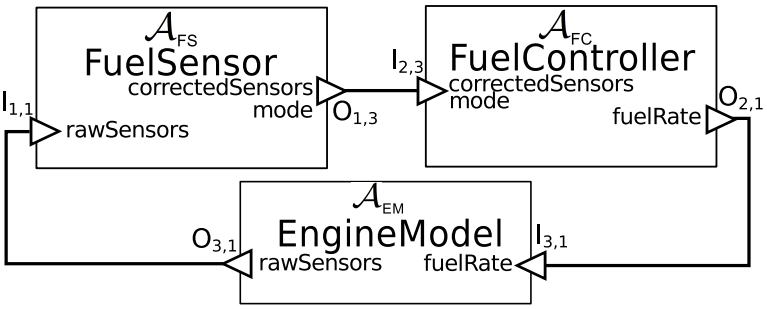


Figure 33: Fuel-Rate-Control architecture with syntactically combined ports.

E Deadlock Semantics and Checks

Deadlocks like defined in the context of the tool UPPAAL have different semantics compared to a deadlock like used in this work. Following it is briefly discussed what are the differences between a deadlock state in UPPAAL in comparison to a deadlock state according to Def. 12. Further, it is shown how to use UPPAAL for checking deadlock freedom according to the semantics of a deadlock like defined in this work.

In this work a deadlock state of a TIOA $\mathcal{A}$ is defined to be deadlock state s_d that is reachable via a finite trace α_d (called a *deadlock trace*). For this deadlock state s_d holds no outgoing transition exists in the induced timed system $\mathcal{T}_{\mathcal{A}}$. Thus, if a deadlock exists, a state is reachable for that holds no transition leaving this state exists at all. In contrast a deadlock state in UPPAAL is defined to be a state where no outgoing discrete transition (edge) can be taken, also in the case previously a continuous transition is taken. In the context of the considered application domain, systems, subsystems and components exist that are used, e.g., for initialization purpose only. Such components are often able reaching a specific state at a certain point of time where no discrete transition need to be taken (e.g., in case an initialization procedure has finished). According to the semantic of a deadlock like used in

UPPAAL, such a component includes a deadlock by construction. Thus, the definition of a deadlock like used in this work seems to be more natural.

As an example consider Fig. 34 containing the initial location *Init* and the second location *Final* for that no outgoing edge exists. Further the TIOA contains a single clock x and an invariant $x \leq 10$ for location *Init*, forcing the TIOA leaving location *Init* within 10 time units. For the location *Final* no invariant exists. According to the UPPAAL semantic this specific TIOA is in a deadlock state when being in location *Final*. This is the case, because no discrete transition (an edge) can be taken when being in location *Final*, no matter if previously a continuous transition is taken in the induced timed system. In contrast, according to the definition of a deadlock state like used in this work (see Def. 12), no deadlock state exists in the TIOA shown in Fig. 34. This is the case because in location *Final* always a discrete transition can be taken allowing time to diverge.

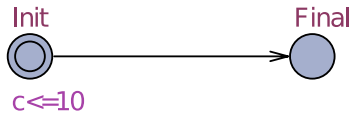


Figure 34: TIOA including a deadlock in UPPAAL.

When using the UPPAAL verifier these semantically differences lead to the limitation that the mechanisms provided by the tool for detecting deadlocks cannot directly be used. UPPAAL provides the keyword *deadlock* for allowing formulating generic statements for checking deadlock freedom according to the UPPAAL semantic. As a result the keyword *deadlock* cannot be used for checking deadlock freedom like defined in this work.

Following we show how to add an additional TA called an *Observer*, allowing deciding if a trace exists that allows time to diverge. Such an observer is shown in Fig. 35. It consists of the two locations *Init* and *Alive* that are connected by two edges in a cyclic form. The TA *Observer* is allowed to stay in the location *Init* for exactly one time unit (realized by the invariant $alive \leq 1$ of location *Init* over the clock $alive$, in combination with the guard $alive \geq 1$ of the only outgoing edge) before being required taking the edge to location *Alive*. Location *Alive* is an urgent location where time is not allowed to pass before the edge back to location *Init* need to be taken. When taking this edge, clock $alive$ is reset to 0 when entering location *Init* where the overall behavior of the TA is repeated.

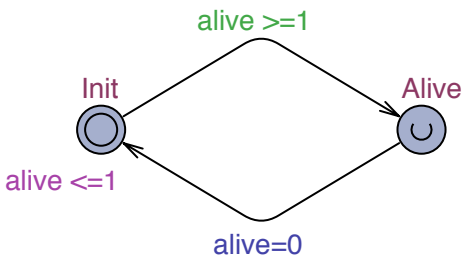


Figure 35: *Observer*.

When searching for deadlocks, e.g., for the TA $\mathcal{A}_{FS}$, the *Observer* TA is added in form of the parallel product $\mathcal{A}_{FS} \parallel$ *Observer*. Like also described in [3] such an *Observer* can be used for checking if the progress of time is always possible. This is achieved by checking for the liveness property, requiring that location *Alive* can be reached on each and every trace of the induced timed system infinitely often within the parallel product. Location *Alive* cannot be reached infinitely often if a path exists where time is not able to diverge. This can be the case if first, a zeno trace exists, not allowing time to diverge, or second, if a deadlock state is reachable where no transition at all can be taken. In this work a TA, and also a TIOA is defined to be valid, iff, it is deadlock and zeno free. As a consequence if time is not able to diverge and location *Alive* of the *Observer* TA cannot be reached infinitely often on at least one path, the TA/TIOA is not valid. Thus, the observer allows checking if a TA is valid, excluding deadlocks and zeno behavior.

Fortunately the tool UPPAAL supports checking liveness properties, allowing deciding if a given TA/TIOA is valid by using the observer shown in Fig. 35. The liveness property for the observer for checking if location *Alive* is reachable on each and every trace infinitely often can simply be formulated as follows: $Init \rightarrow Alive$. This statement can be used in UPPAAL for checking if time is always able to diverge, what excludes deadlock states as well as zeno traces like defined in this work. The same holds for the tool UPPAAL TIGA where liveness properties can be used as a goal allowing searching for strategies avoiding undesired behavior in form of zeno or deadlock behavior, according to the semantics used in this work.

In this work the observer as well as the liveness property is not shown in the discussed examples to allow a better understanding. All investigated deadlocks shown in the examples are cases where a state is reachable allowing not taking a discrete transition, what also is a deadlock according to UPPAAL semantic. Nevertheless, the observer can be used to investigate for zeno and deadlock behavior like defined in our reasoning scheme.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
64	978-3-86956-217-9	**Cyber-Physical Systems with Dynamic Structure: Towards Modeling and Verification of Inductive Invariants**	Basil Becker, Holger Giese
63	978-3-86956-204-9	**Theories and Intricacies of Information Security Problems**	Anne V. D. M. Kayem, Christoph Meinel (Eds.)
62	978-3-86956-212-4	**Covering or Complete? Discovering Conditional Inclusion Dependencies**	Jana Bauckmann, Ziawasch Abedjan, Ulf Leser, Heiko Müller, Felix Naumann
61	978-3-86956-194-3	**Vierter Deutscher IPv6 Gipfel 2011**	Christoph Meinel, Harald Sack (Hrsg.)
60	978-3-86956-201-8	**Understanding Cryptic Schemata in Large Extract-Transform-Load Systems**	Alexander Albrecht, Felix Naumann
59	978-3-86956-193-6	**The JCop Language Specification**	Malte Appeltauer, Robert Hirschfeld
58	978-3-86956-192-9	**MDE Settings in SAP: A Descriptive Field Study**	Regina Hebig, Holger Giese
57	978-3-86956-191-2	**Industrial Case Study on the Integration of SysML and AUTOSAR with Triple Graph Grammars**	Holger Giese, Stephan Hildebrandt, Stefan Neumann, Sebastian Wätzoldt
56	978-3-86956-171-4	**Quantitative Modeling and Analysis of Service-Oriented Real-Time Systems using Interval Probabilistic Timed Automata**	Christian Krause, Holger Giese
55	978-3-86956-169-1	**Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium**	Peter Tröger, Andreas Polze (Eds.)
54	978-3-86956-158-5	**An Abstraction for Version Control Systems**	Matthias Kleine, Robert Hirschfeld, Gilad Bracha
53	978-3-86956-160-8	**Web-based Development in the Lively Kernel**	Jens Lincke, Robert Hirschfeld (Eds.)
52	978-3-86956-156-1	**Einführung von IPv6 in Unternehmensnetzen: Ein Leitfaden**	Wilhelm Boeddinghaus, Christoph Meinel, Harald Sack
51	978-3-86956-148-6	**Advancing the Discovery of Unique Column Combinations**	Ziawasch Abedjan, Felix Naumann
50	978-3-86956-144-8	**Data in Business Processes**	Andreas Meyer, Sergey Smirnov, Mathias Weske
49	978-3-86956-143-1	**Adaptive Windows for Duplicate Detection**	Uwe Draisbach, Felix Naumann, Sascha Szott, Oliver Wonneberg
48	978-3-86956-134-9	**CSOM/PL: A Virtual Machine Product Line**	Michael Haupt, Stefan Marr, Robert Hirschfeld
47	978-3-86956-130-1	**State Propagation in Abstracted Business Processes**	Sergey Smirnov, Armin Zamani Farahani, Mathias Weske

ISBN 978-3-86956-226-1
ISSN 1613-5652