A Process Algebra for Real-Time Programs[⋆]

Henning Dierks

University of Oldenburg, Department of Computer Science,
P.O.Box 2503, 26111 Oldenburg, Germany,
`dierks@informatik.uni-oldenburg.de`

Abstract. We introduce a generalised notion of a real-time specification language ("GPLC-Automata") that can be translated directly into real-time programs. In order to describe the behaviour of several GPLC-Automata implemented on one machine we introduce composition operators which form a process algebra. We give several algebraic laws and prove that each system is equivalent to a system in a certain normal form. Moreover, we demonstrate how a real-time specification in terms of GPLC-Automata can be decomposed into an untimed part and a timed part.

1 Introduction

In this paper we generalise PLC-Automata, a language to specify real-time systems that was motivated by the experiences we made in the UniForM-project [11] with an industrial partner. This notion has been successfully applied to specify a series of academic and industrial case studies [5].

The name stems from the fact that PLC-Automata are compilable into executable code for "Programmable Logic Controllers" (PLC), the hardware target of the project. For formal reasoning we presented in [4] (resp. [7]) a semantics in terms of Duration Calculus [20, 9], an interval based temporal logic, and in terms of Timed Automata [2].

These PLCs are very often used in practice to implement real-time systems. The reason is that they provide both convenient methods to deal with time and an automatic polling mechanism. Nevertheless, every computer system can be used to implement the proposed language if a comparable handling of time and an explicit polling is added.

The process algebra we introduce in this paper allows us to compose generalised PLC-Automata ("GPLC-Automata") which are intended to be implemented on the same PLC. In this case the automata are synchronised in a certain way by the PLC and the process algebra gives us means to exploit this synchronisation for transformations that preserve the semantics. Main benefits are a normal form and a decomposition of a system into its timed and untimed parts. Note that the generalisation of PLC-Automata to GPLC-Automata is necessary because PLC-Automata are not closed under the composition operator we need.

[⋆] This research was partially supported by the Leibniz Programme of the German Research Council (DFG) under grant Ol 98/1-1.

The main difference between the algebra proposed in this paper and existing real-time process algebras like Timed CSP [3, 17] is that our algebra models real-time *programs* including the reaction time of the executing hardware in contrast to assumptions that synchronisation and communication may take place in 0 time units.

2 Programmable Logic Controllers

One goal of the UniForM-project [11] was to verify source code for Programmable Logic Controllers (PLCs). This hardware is often used in industrial practice to control real-time systems like production cells and it can be viewed as a simple computer with a special real-time operating system. PLCs have features for making the design of time- and safety-critical systems easier:

- PLCs have external input and output busses where sensors and actuators can be plugged in.
- PLCs behave in a cyclic manner. Each cycle consists of the following three phases:

  Polling: In this phase the external input bus of the PLC is read and the result is stored in special registers of the PLC. This phase is under the control of the PLC's operating system.

  Computing: After this, the operating system of the PLC executes the user's program once. At this point, the program can make arbitrary computations on the memory including the special registers for the input read. Furthermore, it can change the values in special registers for the output busses of the PLC.

  Moreover, the program can use "timers" which are under the control of the operating system. The program is allowed to set these timers with time values and to check whether this time has elapsed or not. To this end dedicated commands are part of all programming languages for PLCs.

  Updating: In this final phase the special registers for the output busses are read by the operating system. The read values become visible on the output busses.

  The repeated execution of this cycle is managed by the operating system. The only part the programmer has to adapt is the computing phase.
- Depending on the program and on the number of inputs and outputs there is an upper time bound for a cycle that can be used to calculate an upper time bound for the reaction time.
- Convenient standardised libraries are given for the programming languages to simplify the handling of time.

The time consumption of each cycle depends on the duration of actions of the operating system for the polling and updating phases and the duration of the execution of the user's program. The duration of the operating system's actions depends on the number of input and output busses whereas the duration of the program execution can consume an arbitrary amount of time.

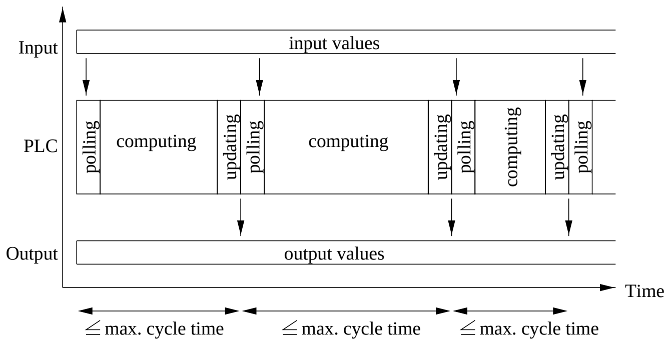


Fig. 1. Cyclic behaviour of a PLC.
A cycle of a PLC consists of three phases: polling, computing, and updating. In our setting we postulate an upper time bound for each cycle.

3 The Definition of GPLC-Automata

In this section we introduce an abstract notion of programs for PLCs. It is a generalisation of PLC-Automata [4] which have been proposed in the UniForM project [11] in order to serve as a common basis for computer scientists and engineers. The idea is that we use extended state-transition diagrams to describe the real-time behaviour of a PLC program. An example of these generalised PLC-Automata is given in Fig. 2

The purpose of this controller is to determine the behaviour of the gas burner. It starts in a state called "idle"; the output "out" is set to the value "*id*" initially. This state holds as long as the polling yields that a Boolean input "*hr*" (standing for heat request) is false. If the polling of the inputs produces "$hr=$ true ", then the system has to switch to the state "purge". When "purge" is entered a timer called "t1" is set to the value of 30 seconds. This means that the Boolean variable "t1" is true for the first 30 seconds after starting and false afterwards. Therefore the system will stay in state "purge" for at least 30 seconds. A timer is equipped with a set of states where it is activated ("activation region"). That means if the system switches into the activation region, then the timer is started. Within the activation region it is allowed to read the value of the timer variable.

The purpose of the "purge" state is to introduce a mandatory delay between attempts of ignition. If such a delay is not present it could happen that several failed attempts of ignition produce a dangerous concentration of gas. When the "purge" state is left, the output is changed to "*ig*". The state "ignite" holds for at least one second due to timer "t2" and the definition of the transitions. If state "burn" is entered the output is changed to "*bn*" and this state holds as long as the polling of input yields that there is still a request for heat ("*hr*") and the Boolean input "*fl*" (standing for "flame") indicates that a flame is present.

Figure 2 contains the equality $\varepsilon = 0.2$ s. This restricts the upper bound of the cycle of the executing hardware. That means the machine has to execute a cycle

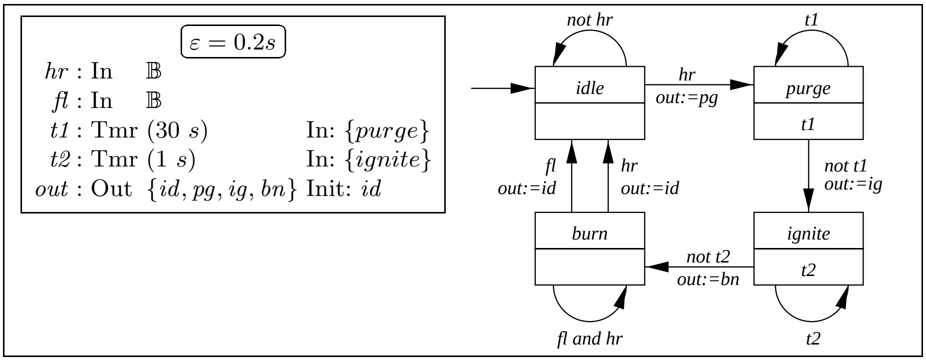


Fig. 2. A gas burner controller as GPLC-Automaton.

within the 0.2 seconds. In the following we will formalise the notion presented in Fig. 2 after some preliminaries.

Let *Var* be the set of all *variables* and assume that each $v \in Var$ has a finite *type* t_v associated with. With $BVar \subseteq Var$ we denote the set of all variables with *Boolean* type.

Let $V, W \subseteq Var$. We call a function *val* that assigns to each $v \in V$ a value of type t_v a *valuation* of V. We use $\mathcal{V}(V)$ to denote the set of *all valuations* of V. If $V \cap W = \varnothing$, $val \in \mathcal{V}(V)$, and $val' \in \mathcal{V}(W)$ we define the *composed valuation* $val \oplus val' \in \mathcal{V}(V \cup W)$ as follows:

$$(val \oplus val')(v) \overset{\text{df}}{=} \begin{cases} val(v), & \text{if } v \in V \\ val'(v), & \text{if } v \in W \end{cases}$$

A generalised PLC-Automaton is in principle an automaton with a finite number of states and several typed variables which are either input, local, or output. Moreover, timer variables (of type Boolean) are allowed. To each timer we assign a running time and a set of states where the timer is active. An upper bound for the cycle time is also included.

Definition 1 (GPLC-Automaton). *A* generalised PLC-Automaton (GPLC-Automaton) *is a structure* $\mathcal{G} = (Q, \Sigma, L, T, \Omega, \longrightarrow, g_0, \varepsilon, \Xi, \Theta)$ *where*

- Q *is a nonempty, finite set of* states*.*
- $\Sigma \subseteq Var$ *is a finite set of* input variables*.*
- $L \subseteq Var$ *is a finite set of* local variables*.*
- $T \subseteq BVar$ *is a finite set of* timer variables*.*
- $\Omega \subseteq Var$ *is a finite set of* output variables*.*
- $\longrightarrow$ *is a function of type* $Q \times \mathcal{V}(L \cup \Omega) \times \mathcal{V}(\Sigma) \times \mathcal{V}(T) \longrightarrow 2^{Q \times \mathcal{V}(L \cup \Omega)} \setminus \{\varnothing\}$ *that describes the* transition relation*.*
- $\varnothing \neq g_0 \subseteq Q \times V(L \cup \Omega)$ *is the* initial condition*, which restricts the initial state and the initial values of the local variables and output variables.*
- $\varepsilon > 0$ *is the* upper time bound *for a cycle.*
- Ξ *is a function of type* $T \longrightarrow 2^Q$ *assigning to each timer variable a set of states, where this timer is activated (*activation function*), and*

– *Θ is a function of type $T \longrightarrow \mathbb{R}_{>0}$ assigning to each timer variable a* running time.

Furthermore Σ, L, T, and Ω are disjoint and the well-formedness condition

$$\forall q \in Q, v \in \mathcal{V}(L \cup \Omega), \varphi \in \mathcal{V}(\Sigma), \tau_1, \tau_2 \in \mathcal{V}(T) : \tag{1}$$
$$\tau_1|_{\{t \in T | q \in \Xi(t)\}} = \tau_2|_{\{t \in T | q \in \Xi(t)\}} \Longrightarrow \longrightarrow (q, v, \varphi, \tau_1) = \longrightarrow (q, v, \varphi, \tau_2)$$

has to hold which says that timer which are not active cannot influence the behaviour of the system.

We will use the notation $(q, v) \xrightarrow{\varphi, \tau}_{\mathcal{G}} (q', v')$ for $q, q' \in Q$, $v, v' \in \mathcal{V}(L \cup \Omega)$, $\varphi \in \mathcal{V}(\Sigma)$, and $\tau \in \mathcal{V}(T)$ if $(q', v') \in \longrightarrow_{\mathcal{G}} (q, v, \varphi, \tau)$.

Note that we allow a *set* of initial states and do not need a notion of final states since the polling systems are intended to run infinitely. The definition of the transition relation postulates that there is at least one allowed reaction but there may be several choices. In case that the transition relation allows more than one transition the system may choose a transition nondeterministically.

The Boolean timer variables T can be used to measure the time. If $\mathit{tmr} \in T$ is a timer variable with *activation region* $\Xi(\mathit{tmr}) \subseteq Q$, then it carries the value true if the systems stayed less than $\Theta(\mathit{tmr})$ seconds in the activation region. It carries the value false if $\Theta(\mathit{tmr})$ seconds have elapsed in the activation region.

Note that in Fig. 2 a syntax for both guards and actions annotated to transitions is given. Here we omit the formal definition of syntax and semantics of both since they are straightforward.

The operational behaviour of a GPLC-Automaton as in the definition above is as follows: in each cycle the system stores the polled input values in the variables of Σ. If there are more than one possible transition for the current state, the current valuation of the input variable, timer variables, local variables, and output variables, then the system chooses nondeterministically one of these transitions. Finally, the cycle is finished and the values of the output variables become visible from the outside.

In [5] it is shown how to generate systematically executable source code from GPLC-Automata for PLCs. The semantics that will be presented in the following section describes the behaviour of the PLC executing the source code within the given upper bound for the cycle time. The analysis whether a given PLC is able to execute the source code generated is not difficult since no loops or jumps are necessary for the implementation.

4 The Timed Automaton Semantics of GPLC-Automata

In this section we present an operational semantics of GPLC-Automata in terms of Timed Automata. For the definition of Timed Automata the reader is referred to App. A. We first present the components of the Timed Automaton $\mathcal{T}(\mathcal{G})$ that is associated to a given GPLC-Automaton $\mathcal{G}$, and then give some intuition.

Each location[1] of $\mathcal{T}(\mathcal{G})$ is a 6-tuple $(i, \varphi, \phi, q, \pi, \tau)$, where

$i \in \{0, 1, 2\}$ describes the current status of the PLC ("program counter"),
$\varphi \in \mathcal{V}(\Sigma)$ contains the current input valuation,
$\phi \in \mathcal{V}(\Sigma)$ contains the last input valuation that has been polled,
$q \in Q$ is the current state of the GPLC-Automaton,
$\pi \in \mathcal{V}(L \cup \Omega)$ is the current valuation of local and output variables, and
$\tau \in \mathcal{V}(T)$ is the last timer valuation that has been tested.

There are three kinds of clocks in use:

x measures the time for which the current latest input valuation is stable,
y_t measures the time that has elapsed since the latest start of timer t, and
z measures the time elapsed in the current cycle of the PLC.

The idea of the program counter for GPLC-Automata is to model the internal status of the polling device. If the program counter is 0, then the polling has not happened in the current cycle. The change from 0 to 1 ((GTA-2) in Table 1) represents the polling of the system and hence we copy the second component of the location to the third. This is not allowed if the current input valuation has changed at the same point of time. Therefore we have to test whether the x-clock is greater than 0. This clock is reset whenever the environment changes the input valuation (GTA-1).

If the program counter is 1, then the polling has happened and the computation takes place. However, the *result* of the computation will not be visible before the cycle ends (cf. Fig. 1). For the semantics it is important which valuation of the timer variables was valid when the computing phase took place. Therefore, we will record this valuation in the sixth component in the location of the Timed Automaton (GTA-3). If the program counter has value 2, then the computation is finished and the updating phase takes place. When this phase is left and the counter is set to 0 again, the cycle ends and we change the state, local variables, and output variables accordingly (GTA-4).

Definition 2 (Timed Automata semantics of GPLC-Automata). *Let $\mathcal{G} = (Q, \Sigma, L, T, \Omega, \longrightarrow, g_0, \varepsilon, \Xi, \Theta)$ be a GPLC-Automaton. We define $\mathcal{T}(\mathcal{G})$ to be the Timed Automaton $(\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{IV}, \mathcal{P}, \mu, S_0)$ with*

- $\mathcal{S} \overset{df}{=} \{0, 1, 2\} \times \mathcal{V}(\Sigma) \times \mathcal{V}(\Sigma) \times Q \times \mathcal{V}(L \cup \Omega) \times \mathcal{V}(T)$ *as* locations*,*
- $\mathcal{X} \overset{df}{=} \{x, z\} \cup \{y_t | t \in T\}$ *as* clocks*,*
- $\mathcal{L} \overset{df}{=} \mathcal{V}(\Sigma) \cup \{poll, test, tick\}$ *as* labels*,*
- *the set of* edges $\mathcal{E}$ *is given in Table 1,*
- $\mathcal{IV}(s) \overset{df}{=} z \leq \varepsilon$ *as* invariant *for each location* $s \in \mathcal{S}$*,*
- $\mathcal{P} \overset{df}{=} \{obs = v | obs \in \Sigma \cup L \cup \Omega\}, v \in t_{obs}\}$ *as the set of* propositions*,*

[1] Note that the notion "locations" refers to the Timed Automaton and the notion "states" to the GPLC-Automaton.

$$(i,\varphi,\phi,q,\pi,\tau) \xrightarrow{\varphi',\mathsf{true},\{x\}} (i,\varphi',\phi,q,\pi,\tau) \qquad \text{(GTA-1)}$$

$$(0,\varphi,\phi,q,\pi,\tau) \xrightarrow{poll,0<x\wedge 0<z,\varnothing} (1,\varphi,\varphi,q,\pi,\tau) \qquad \text{(GTA-2)}$$

$$(1,\varphi,\phi,q,\pi,\tau) \xrightarrow{test,\mathcal{C}(q,\tau'),\varnothing} (2,\varphi,\phi,q,\pi,\tau') \qquad \text{(GTA-3)}$$

$$(2,\varphi,\phi,q,\pi,\tau) \xrightarrow{tick,\mathsf{true},\mathcal{RS}(q,q')\cup\{z\}} (0,\varphi,\phi,q',\pi',\tau) \qquad \text{(GTA-4)}$$

$$\text{with } (q,\pi) \xrightarrow{\phi,\tau}_{\mathcal{G}} (q',\pi')$$

Table 1. Transitions of the Timed Automaton $\mathcal{T}(\mathcal{G})$.
with $i \in \{0,1,2\}$, $\varphi,\varphi',\phi \in \mathcal{V}(\Sigma)$, $q,q' \in Q$, $\pi,\pi' \in \mathcal{V}(L\cup\Omega)$, $\tau,\tau' \in \mathcal{V}(T)$,

$$\mathcal{C}(q,\tau) \overset{\text{df}}{=} \bigwedge_{\substack{q\in\Xi(t)\\ \tau(t)}} y_t < \Theta(t) \wedge \bigwedge_{\substack{q\in\Xi(t)\\ \neg\tau(t)}} y_t \geq \Theta(t), \text{ and}$$

$$\mathcal{RS}(q,q') \overset{\text{df}}{=} \{y_t | t\in T, q'\in\Xi(t), q\notin\Xi(t)\}$$

- $\mu(i,\varphi,\phi,q,\pi,\tau) \overset{df}{=} \{\sigma=\varphi(\sigma)|\sigma\in\Sigma\} \cup \{obs=\pi(obs)|obs\in L\cup\Omega\}$ *as the propositions for each location* $(i,\varphi,\phi,q,\pi,\tau)\in\mathcal{S}$,
- $S_0 \overset{df}{=} \{(0,\varphi,\phi,q,\pi,\tau)\in\mathcal{S}|(q,\pi)\in g_0\}$ *as set of* initial locations.

In [5] an alternative semantics in terms of the interval based temporal logic Duration Calculus [20, 9] was presented. In contrast to the Timed Automaton semantics above which is of operational kind, the Duration Calculus semantics for GPLC-Automata is of denotational kind. It was shown that both semantics are equivalent in an appropriate sense. The advantage of the Timed Automaton semantics is that it is easier to understand and that Model-Checkers for Timed Automata are applicable to GPLC-Automata. The advantage of the Duration Calculus semantics is that logical reasoning is easier. In [6] a synthesis procedure for GPLC-Automata was presented and proved to be correct with the help of the logical semantics.

5 A Process Algebra for GPLC-Automata

In this section we present operators to compose GPLC-Automata which are implemented *on the same PLC*. The most important operator is the sequential composition ("$\mathcal{G}_1;\mathcal{G}_2$"). The name stems from the fact that the computation of the composition is done by the computation of $\mathcal{G}_1$ first and second the computation of $\mathcal{G}_2$ in the same cycle of the implementing PLC (cf. Fig. 3). Note that we assume that the cycle time of a composition is at least as fast as the cycle times of the composed component. The reason is that in this case we get the property that an automaton $\mathcal{G}$ in sequential composition with an arbitrary automaton is still a refinement of the $\mathcal{G}$.

When we define the sequential composition we have to consider that inputs of one automaton which are triggered by outputs of the other automaton have to be handled in a different way than usual inputs. The reason is that polling is

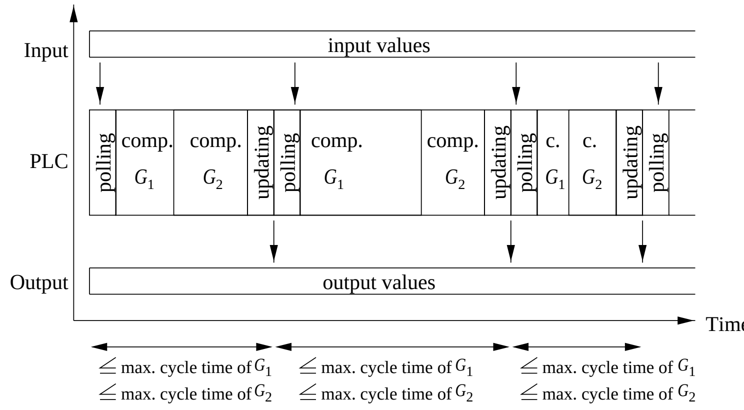

Input
input values
PLC
polling
comp. G_1
comp. G_2
updating
polling
comp. G_1
comp. G_2
updating
polling
c. G_1
c. G_2
updating
polling
Output
output values
Time
≤ max. cycle time of G_1
≤ max. cycle time of G_2
≤ max. cycle time of G_1
≤ max. cycle time of G_2
≤ max. cycle time of G_1
≤ max. cycle time of G_2

Fig. 3. Cyclic behaviour of $\mathcal{G}_1;\mathcal{G}_2$.

not necessary because the information is always present in the PLC. Therefore, we have three cases of inputs for the sequential composition $\mathcal{G}_1;\mathcal{G}_2$:

- the input is not an output of the other automaton. In this case the input values are determined by the polling phase.
- the input of $\mathcal{G}_1$ is triggered by an output of $\mathcal{G}_2$. Here the input value is determined by the corresponding output value of $\mathcal{G}_2$ in the *previous* cycle.
- the input of $\mathcal{G}_2$ is triggered by an output of $\mathcal{G}_1$. In this case the input value is determined by the corresponding output value of $\mathcal{G}_1$ in the *current* cycle.

Definition 3 (Sequential composition). *Let $\mathcal{G}_i$ be GPLC-Automata with $\mathcal{G}_i = (Q_i, \Sigma_i, L_i, T_i, \Omega_i, \longrightarrow_i, g_{0,i}, \varepsilon_i, \Xi_i, \Theta_i)$ for $i = 1,2$ and disjoint variable sets $L_1, L_2, T_1, T_2, \Omega_1, \Omega_2$ and $\Sigma_i \cap (L_{3-i} \cup T_{3-i}) = \varnothing$ for $i = 1,2$. We call the GPLC-Automaton*

$$\mathcal{G}_1;\mathcal{G}_2 = (Q_1 \times Q_2, \Sigma, L_1 \cup L_2, T_1 \cup T_2, \Omega_1 \cup \Omega_2, \longrightarrow, g_0, \varepsilon, \Xi, \Theta)$$

the sequential composition of $\mathcal{G}_1$ and $\mathcal{G}_2$ *iff*

$$\begin{aligned}
\Sigma &= (\Sigma_1 \cup \Sigma_2) \setminus (\Omega_1 \cup \Omega_2) \\
\longrightarrow ((q_1, q_2), v, \varphi, \tau) &= \{((q_1', q_2'), v') \in Q \times \mathcal{V}(L \cup \Omega) | \\
&\qquad (q_1, v|_{L_1 \cup \Omega_1}) \xrightarrow{(\varphi \oplus v)|_{\Sigma_1}, \tau|_{T_1}}{}_1 (q_1', v'|_{L_1 \cup \Omega_1}) \wedge \\
&\qquad (q_2, v|_{L_2 \cup \Omega_2}) \xrightarrow{(\varphi \oplus v')|_{\Sigma_2}, \tau|_{T_2}}{}_2 (q_2', v'|_{L_2 \cup \Omega_2}) \\
g_0 &= \{((q_1, q_2), v) \in Q \times \mathcal{V}(L \cup \Omega) | \forall i : (q_i, v|_{L_i \cup \Omega_i}) \in g_{0,i}\} \\
\varepsilon &= \min\{\varepsilon_i | i = 1, 2\} \\
\Xi(tmr) &= \begin{cases} \Xi_1(tmr) \times Q_2, & \text{if } tmr \in T_1 \\ Q_1 \times \Xi_2(tmr), & \text{if } tmr \in T_2 \end{cases} \\
\Theta &= \Theta_1 \oplus \Theta_2
\end{aligned}$$

This definition of sequential composition assumes that the interface between the automata is given by the sets $\Sigma_1 \cap \Omega_2$ (inputs of $\mathcal{G}_1$ connected with outputs of $\mathcal{G}_2$) and $\Sigma_2 \cap \Omega_1$ (inputs of $\mathcal{G}_2$ connected with outputs of $\mathcal{G}_1$). This concept of an interface definition by equality of names may be too weak. Moreover, we cannot express that we want to connect an input with an output of the same automaton or that we want to hide an output and conceive it as local.

Hence, we define ways to change the interface, i.e., the set of inputs and the set of outputs. First, we introduce renaming of inputs where it is possible to replace the current names of the inputs by other names. It is even possible to choose locals and outputs as new name for inputs. In this case the input is removed from the external interface.

Definition 4 (Renaming of inputs). *Consider a GPLC-Automaton $\mathcal{G} = (Q, \Sigma, L, T, \Omega, \longrightarrow, g_0, \varepsilon, \Xi, \Theta)$ and a function f with domain Σ which is type consistent ($\forall \sigma \in \Sigma : t_\sigma = t_{f(\sigma)}$) and which does not map into the timers ($f(\Sigma) \cap T = \varnothing$). Then we call the automaton*

$$f{:}\mathcal{G} \overset{df}{=} (Q, f(\Sigma) \setminus (L \cup \Omega), L, T, \Omega, \longrightarrow_f, g_0, \varepsilon, \Xi, \Theta)$$

the automaton $\mathcal{G}$ renamed by f *where*

$$(q, v) \xrightarrow{\varphi,\tau}_f (q', v') \quad \textit{iff} \quad (q, v) \xrightarrow{(\varphi \oplus v) \circ f, \tau} (q', v')$$

With the next definition we allow to remove outputs from the interface. This transformation is called “hiding”. The hidden outputs are shifted into the set of locals:

Definition 5 (Hiding of outputs).
Let $\mathcal{G} = (Q, \Sigma, L, T, \Omega, \longrightarrow, g_0, \varepsilon, \Xi, \Theta)$ be a GPLC-Automaton and $H \subseteq \Omega$. We call the automaton

$$\mathcal{G} \setminus H \overset{df}{=} (Q, \Sigma, L \cup H, T, \Omega \setminus H, \longrightarrow, g_0, \varepsilon, \Xi, \Theta)$$

the automaton $\mathcal{G}$ with hidden outputs H.

In the following we examine some properties of the operations defined above. To this end we define a notion of refinement and equivalence in usual manner. This definition employs a notion of simulation for Timed Automata that is defined in App. A.

Definition 6 (Refinement and equivalence). *Let $\mathcal{G}_1, \mathcal{G}_2$ be GPLC-Automata with $\Sigma_1 \cup L_1 \cup \Omega_1 \supseteq \Sigma_2 \cup L_2 \cup \Omega_2$ and $L_1 \cup \Omega_1 \supseteq L_2 \cup \Omega_2$. We say that $\mathcal{G}_1$* is a refinement of $\mathcal{G}_2$ *(in symbols: $\mathcal{G}_1 \Rrightarrow \mathcal{G}_2$) iff*

$$\mathcal{T}(\mathcal{G}_1) \lesssim_\beta \mathcal{T}(\mathcal{G}_2)$$

with

$$\begin{aligned}\beta(\{obs = val(obs) \mid obs \in \Sigma_1 \cup L_1 \cup \Omega_1\}) \\ = \{obs = val(obs) \mid obs \in \Sigma_2 \cup L_2 \cup \Omega_2\}\end{aligned}$$

for each valuation $val \in \mathcal{V}(\Sigma_1 \cup L_1 \cup \Omega_1)$.

If $\mathcal{G}_1 \Rrightarrow \mathcal{G}_2$ and $\mathcal{G}_2 \Rrightarrow \mathcal{G}_1$ hold we say that $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent. *In symbols: $\mathcal{G}_1 \equiv \mathcal{G}_2$. It is clear how to lift $\Rrightarrow$ and $\equiv$ to composed systems.*

Note that equivalence of $\mathcal{G}_1$ and $\mathcal{G}_2$ implies $\Sigma_1 = \Sigma_2$.

Lemma 1 (Algebraic rules). *Let $\mathcal{G}, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ be GPLC-Automata. It holds:*

Associativity of sequential composition:

$$\mathcal{G}_1; (\mathcal{G}_2; \mathcal{G}_3) \equiv (\mathcal{G}_1; \mathcal{G}_2); \mathcal{G}_3 \tag{2}$$

Monotonicity: ($\Sigma_{\mathcal{G}_1} = \Sigma_{\mathcal{G}_2}$, $L_{\mathcal{G}_1} = L_{\mathcal{G}_2}$, $T_{\mathcal{G}_1} = T_{\mathcal{G}_2}$, $\Omega_{\mathcal{G}_1} = \Omega_{\mathcal{G}_2}$)

$$\mathcal{G}_1 \equiv \mathcal{G}_2 \Longrightarrow \mathcal{G}_1 \setminus H \equiv \mathcal{G}_2 \setminus H \quad \textit{provided that } \mathcal{G}_1 \setminus H \textit{ is legal} \tag{3}$$

$$\Longrightarrow f\!:\!\mathcal{G}_1 \equiv f\!:\!\mathcal{G}_2 \quad \textit{provided that } f\!:\!\mathcal{G}_1 \textit{ is legal} \tag{4}$$

$$\Longrightarrow \mathcal{G}_1; \mathcal{G} \equiv \mathcal{G}_2; \mathcal{G} \quad \textit{provided that } \mathcal{G}_1; \mathcal{G} \textit{ is legal} \tag{5}$$

$$\Longrightarrow \mathcal{G}; \mathcal{G}_1 \equiv \mathcal{G}; \mathcal{G}_2 \quad \textit{provided that } \mathcal{G}; \mathcal{G}_1 \textit{ is legal} \tag{6}$$

Commutativity of sequential composition:

$$\mathcal{G}_1; \mathcal{G}_2 \equiv \mathcal{G}_2; \mathcal{G}_1 \textit{ if } \Sigma_{\mathcal{G}_1} \cap \Omega_{\mathcal{G}_2} = \Sigma_{\mathcal{G}_2} \cap \Omega_{\mathcal{G}_1} = \varnothing \tag{7}$$

Refinement by sequential composition:

$$\mathcal{G}_1; \mathcal{G}_2 \Rrightarrow \mathcal{G}_1 \tag{8}$$

$$\mathcal{G}_1; \mathcal{G}_2 \Rrightarrow \mathcal{G}_2 \tag{9}$$

Concatenation of renamings: ($\mathrm{dom}(f) = \Sigma_{\mathcal{G}}$, $\mathrm{dom}(f') = \Sigma_{f\mathcal{G}}$*)*

$$f'\!:\!(f\!:\!\mathcal{G}) \equiv f''\!:\!\mathcal{G} \textit{ with } f''(\sigma) = \begin{cases} f'(f(\sigma)), & \textit{if } f(\sigma) \in \mathrm{dom}(f') \\ f(\sigma), & \textit{if } f(\sigma) \notin \mathrm{dom}(f') \end{cases} \tag{10}$$

Concatenation of hidings: ($H \subseteq \Omega_{\mathcal{G}}$, $H' \subseteq \Omega_{\mathcal{G}} \setminus H$)

$$(\mathcal{G} \setminus H) \setminus H' \equiv \mathcal{G} \setminus (H \cup H') \tag{11}$$

Hiding and sequential composition: ($H_i \subseteq \Omega_{\mathcal{G}_i}$, $H \subseteq \Omega_{\mathcal{G}_1} \cup \Omega_{\mathcal{G}_2}$)

$$(\mathcal{G}_1 \setminus H_1); \mathcal{G}_2 \equiv (\mathcal{G}_1; \mathcal{G}_2) \setminus H_1 \textit{ where } H_1 \cap \Sigma_{\mathcal{G}_2} = \varnothing \tag{12}$$

$$\mathcal{G}_1; (\mathcal{G}_2 \setminus H_2) \equiv (\mathcal{G}_1; \mathcal{G}_2) \setminus H_2 \textit{ where } H_2 \cap \Sigma_{\mathcal{G}_1} = \varnothing \tag{13}$$

$$(\mathcal{G}_1; \mathcal{G}_2) \setminus H \equiv ((\mathcal{G}_1 \setminus (H \setminus \Sigma_{\mathcal{G}_2})); (\mathcal{G}_2 \setminus (H \setminus \Sigma_{\mathcal{G}_1}))) \setminus (H \cap (\Sigma_{\mathcal{G}_1} \cup \Sigma_{\mathcal{G}_2})) \tag{14}$$

Hiding and renaming: ($\mathrm{dom}(f) = \Sigma_{\mathcal{G}}$, $H \subseteq \Omega_{\mathcal{G}}$*)*

$$f\!:\!(\mathcal{G} \setminus H) \equiv (f\!:\!\mathcal{G}) \setminus H \tag{15}$$

Renaming and sequential composition: ($\mathrm{dom}(f) = \Sigma_{\mathcal{G}_1;\mathcal{G}_2}$, $\mathrm{dom}(f_i) = \Sigma_{\mathcal{G}_i}$*)*

$$f\!:\!(\mathcal{G}_1; \mathcal{G}_2) \equiv ((\mathsf{id}_{\Sigma_{\mathcal{G}_1} \cap \Omega_{\mathcal{G}_2}} \oplus f|_{\Sigma_{\mathcal{G}_1} \setminus \Omega_{\mathcal{G}_2}})\!:\!\mathcal{G}_1); \\ ((\mathsf{id}_{\Sigma_{\mathcal{G}_2} \cap \Omega_{\mathcal{G}_1}} \oplus f|_{\Sigma_{\mathcal{G}_2} \setminus \Omega_{\mathcal{G}_1}})\!:\!\mathcal{G}_2) \tag{16}$$

$$(f_1\!:\!\mathcal{G}_1); \mathcal{G}_2 \equiv (f_1|_{\Sigma_{\mathcal{G}_1} \setminus f_1^{-1}(\Omega_{\mathcal{G}_2})} \oplus \mathsf{id}_{\Sigma_{\mathcal{G}_2} \setminus \Omega_{\mathcal{G}_1}})\!: \\ (((f_1|_{f_1^{-1}(\Omega_{\mathcal{G}_2})} \oplus \mathsf{id}_{\Sigma_{\mathcal{G}_1} \setminus f_1^{-1}(\Omega_{\mathcal{G}_2})})\!:\!\mathcal{G}_1); \mathcal{G}_2) \tag{17}$$

$$\mathcal{G}_1; (f_2\!:\!\mathcal{G}_2) \equiv (f_2|_{\Sigma_{\mathcal{G}_2} \setminus f_2^{-1}(\Omega_{\mathcal{G}_1})} \oplus \mathsf{id}_{\Sigma_{\mathcal{G}_1} \setminus \Omega_{\mathcal{G}_2}})\!: \\ (\mathcal{G}_1; ((f_2|_{f_2^{-1}(\Omega_{\mathcal{G}_1})} \oplus \mathsf{id}_{\Sigma_{\mathcal{G}_2} \setminus f_2^{-1}(\Omega_{\mathcal{G}_1})})\!:\!\mathcal{G}_2)) \tag{18}$$

Proof. of (8): It is clear from the definition of sequential composition that

$$\begin{aligned}\Sigma_{\mathcal{G}_1;\mathcal{G}_2} \cup L_{\mathcal{G}_1;\mathcal{G}_2} \cup \Omega_{\mathcal{G}_1;\mathcal{G}_2} &= (\Sigma_1 \cup \Sigma_2) \setminus (\Omega_1 \cup \Omega_2) \cup L_1 \cup L_2 \cup \Omega_1 \cup \Omega_2 \\ &\supseteq \Sigma_1 \cup L_1 \cup \Omega_1\end{aligned}$$

Also clear is $L_{\mathcal{G}_1;\mathcal{G}_2} \cup \Omega_{\mathcal{G}_1;\mathcal{G}_2} \supseteq L_1 \cup \Omega_1$. Hence, the requirements of Def. 6 are fulfilled. We have to construct for each run of $\mathcal{T}(\mathcal{G}_1;\mathcal{G}_2)$ a run of $\mathcal{T}(\mathcal{G}_1)$ with the same time stamps and the same propositions. Let $((s_j, v_j, t_j)_{j\in\mathbb{N}_0}) \in \mathcal{R}(\mathcal{T}(\mathcal{G}_1;\mathcal{G}_2))$. Due to the definition of the sequential composition we know that the clock valuations v_j are functions with domain $\{x,z\} \cup \{y_t | t \in T_1 \cup T_2\}$. Let $v'_j = v_j|_{\{x,z\}\cup\{y_t|t\in T_1\}}$ the restriction of v_j to the clocks of $\mathcal{T}(\mathcal{G}_1)$. Due to the definitions we can conclude that $s_j = (i_j, \varphi_j, \phi_j, q_j, \pi_j, \tau_j)$ with $i_j \in \{0,1,2\}$, $\varphi_j, \phi_j \in \mathcal{V}(\Sigma_{\mathcal{G}_1;\mathcal{G}_2})$, $q_j \in Q_1 \times Q_2$, $\pi_j \in \mathcal{V}(L_1 \cup L_2 \cup \Omega_1 \cup \Omega_2)$, and $\tau_j \in \mathcal{V}(T_1 \cup T_2)$. We construct the states s'_j as follows:

$$\begin{aligned}&s'_j = (i_j, (\varphi_j \oplus \pi_j)|_{\Sigma_1}, (\phi_j \oplus \pi'(j))|_{\Sigma_1}, p_1(q_j), \pi_j|_{L_1\cup\Omega_1}, \tau_j|_{T_1}) \\ &\text{with } \pi'(j) = \pi_{\max\{k\in\mathbb{N}_0 | k\le j, i_k=1\}}\end{aligned}$$

It is simple to verify that $((s'_j, v'_j, t_j)_{j\in\mathbb{N}_0})$ is a run of $\mathcal{T}(\mathcal{G}_1)$ with the property

$$\begin{aligned}\beta(\mu_{\mathcal{T}(\mathcal{G}_1;\mathcal{G}_2)}(s_j)) &= \beta(\{obs = val(obs) \mid obs \in \Sigma_{\mathcal{G}_1;\mathcal{G}_2} \cup L_{\mathcal{G}_1;\mathcal{G}_2} \cup \Omega_{\mathcal{G}_1;\mathcal{G}_2}\}) \\ &= \{obs = val(obs) \mid obs \in \Sigma_{\mathcal{G}_1} \cup L_{\mathcal{G}_1} \cup \Omega_{\mathcal{G}_1}\} \\ &= \mu_{\mathcal{T}(\mathcal{G}_1)}(s'_j)\end{aligned}$$

Theorem 1 (Normal form). *Each system that is composed by sequential composition, renaming, and hiding is equivalent to a system of the form*

$$((f_1 : \mathcal{G}_1); \ldots ; (f_n : \mathcal{G}_n)) \setminus H$$

with appropriate GPLC-Automata $\mathcal{G}_1, \ldots, \mathcal{G}_n$, renaming functions $f_1, \ldots, f_n$ and H.

Proof. First note that (10), (11), (12), (13), (15), and (16), are all of the form $\mathcal{G}_{left} \equiv \mathcal{G}_{right}$ with $\Sigma_{left} = \Sigma_{right}$, $L_{left} = L_{right}$, $T_{left} = T_{right}$, and $\Omega_{left} = \Omega_{right}$. Therefore, the requirements of the monotonicity laws (3)–(6) are fulfilled. Hence, the above formulas are applicable to arbitrary subsystems of a composed system. To reach the normal form we first shift all hidings to the outermost position. To achieve this we can apply (12), (13), and (15) as long as possible. Then apply (11) to summarise all hidings in one operation. Then use (16) to shift the renamings to the innermost positions. Equivalence (10) can be applied to reach the normal form we desire.

Theorem 2 (Decomposition of timers). *Let $\mathcal{G}$ be a GPLC-Automaton with $\mathcal{G} = (Q, \Sigma, L, T, \Omega, \longrightarrow, g_0, \varepsilon, \Xi, \Theta)$ and $t \in T$. Then holds $\mathcal{G} \equiv$*

$(Timer_t^{\varepsilon_\mathcal{G}}; \mathcal{G}_{-t}) \setminus \{act, runs\}$ where act and runs are fresh variables of Boolean type. The GPLC-Automaton $\mathcal{G}_{-t}$ is defined as follows:

$$\mathcal{G}_{-t} \overset{df}{=} (Q, \Sigma \cup \{runs\}, L, T \setminus \{t\}, \Omega \cup \{act\}, \longrightarrow', g_0', \varepsilon, \Xi|_{T\setminus\{t\}}, \Theta|_{T\setminus\{t\}})$$

with

$$g_0' \overset{df}{=} \{(q,\pi) | (q, \pi|_{L\cup\Omega}) \in g_0, \pi(act) \iff q \in \Xi(t)\}$$

$$\longrightarrow'(q,\pi,\sigma,\tau) \overset{df}{=} \{(q',\pi') | \pi'(act) \iff q' \in \Xi(t),$$
$$(q, \pi|_{L\cup\Omega}) \xrightarrow{\sigma|_\Sigma, (\tau\oplus(t\mapsto\sigma(t)))\}} (q', \pi'|_{L\cup\Omega}).$$

The automaton $Timer_t^{\varepsilon_\mathcal{G}}$ is given in Fig. 4.

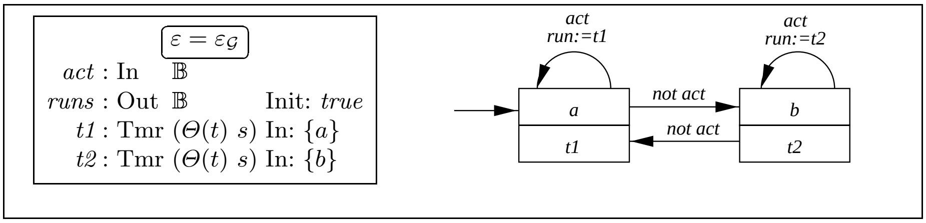

$\varepsilon = \varepsilon_\mathcal{G}$
act : In $\mathbb{B}$
runs : Out $\mathbb{B}$ Init: *true*
t1 : Tmr ($\Theta(t)$ *s*) In: $\{a\}$
t2 : Tmr ($\Theta(t)$ *s*) In: $\{b\}$
act
run:=t1
act
run:=t2
not act
not act
a
t1
b
t2

Fig. 4. Automaton $Timer_t^{\varepsilon_\mathcal{G}}$.

The idea of this decomposition is to replace a timer t of an automaton $\mathcal{G}$ by a Boolean input (*runs*) that is triggered by another automaton ($Timer_t^{\varepsilon_\mathcal{G}}$). The latter gets the information whether the reduced $\mathcal{G}$ is in the activation region of t by the Boolean channel *act*. Since $Timer_t^{\varepsilon_\mathcal{G}}$ is executed before the reduced automaton $\mathcal{G}_{-t}$ is executed, $Timer_t^{\varepsilon_\mathcal{G}}$ has to anticipate that $\mathcal{G}_{-t}$ changes into the activation region the in current cycle. Hence, $Timer_t^{\varepsilon_\mathcal{G}}$ starts in each cycle a timer with the same time as t as long as $\mathcal{G}_{-t}$ is not in the activation region of t.

6 Application Example

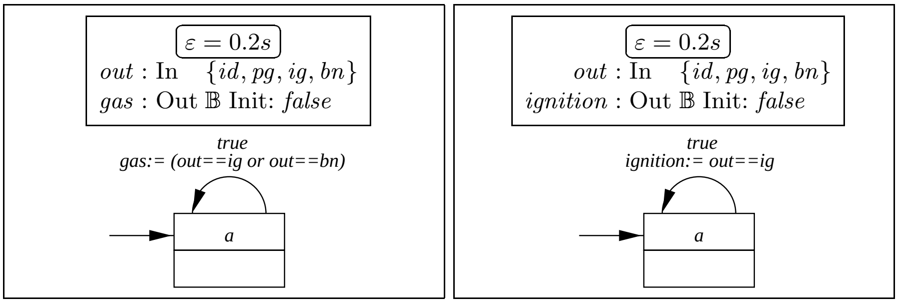
In this section we apply the process algebra to the gas burner GPLC-Automaton *GB* given in Fig. 2. In order to control the gas valve we assume that another automaton ("*gas*") computes the signal for the valve depending on the output of *GB*. This automaton is given in Fig. 5. Analogously we assume that a controller ("*ign*") is given to control the ignition of the flame (cf. Fig. 5).

If we consider an alternative implementation where the variables *ignition* and *gas* are manipulated directly like the way as given in Fig. 6, the question arises whether both ways are equivalent. The answer is positive, ie. in symbols:

$$(GB; ign; gas) \setminus \{out\} \equiv GB' \equiv (GB; gas; ign) \setminus \{out\}.$$

By the decomposition theorem it is also possible to extract a controller which does not use timers:

$$GB' \equiv (\underbrace{Timer_{t1}^{0.2}; Timer_{t2}^{0.2}}_{\text{timed}}; \underbrace{(GB'_{-t1})_{-t2}}_{\text{untimed}}) \setminus \{act_{t1}, act_{t2}, runs_{t1}, runs_{t2}\}$$



$\varepsilon = 0.2s$
out : In $\{id, pg, ig, bn\}$
gas : Out $\mathbb{B}$ Init: $false$
true
gas:= (out==ig or out==bn)
a
$\varepsilon = 0.2s$
out : In $\{id, pg, ig, bn\}$
$ignition$: Out $\mathbb{B}$ Init: $false$
true
ignition:= out==ig
a

Fig. 5. Controller gas **(LHS) and** ign **(RHS).**

7 Conclusion

We introduced several composition operators for GPLC-Automata which are implemented on the same PLC. These definitions were motivated by the observable behaviour of the real machine executing the source code produced from the GPLC-Automata. Due to the equivalences that are proved in this paper we are allowed to transform a system into a normal form and/or into timed and untimed parts.

In the future we plan to exploit these results to improve the model-checking of PLC-Automata. In Moby/PLC [8, 18], a tool suite for PLC-Automata, it is possible to translate a system of PLC-Automata into its Timed Automaton semantics in the syntax of the model-checkers Uppaal [12] and Kronos [19]. With the help of the process algebra given in this paper we expect to improve this translation such that the system become smaller and hence faster to check. The reason is that the current translation of GPLC-Automata systems does not exploit the information whether two automata are in sequential composition. Hence, it introduces for each automaton the clocks x and z. If it would exploit the information it could save two clocks per sequential composition.

Acknowledgements: The author thanks E.-R. Olderog, J. Hoenicke, and other members of the “semantics group” in Oldenburg for detailed comments and various discussions on the subject of this paper. Furthermore, he likes to thank the anonymous referees for the valuable hints to improve the paper.

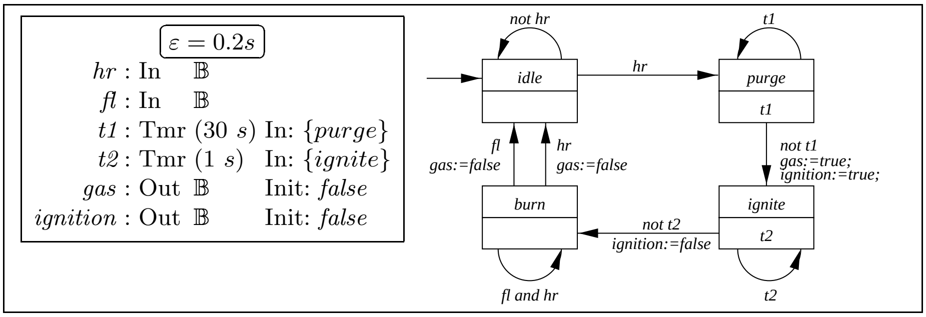


Fig. 6. An alternative controller GB' **for the gas burner.**

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-Checking for Real-Time Systems. In *Fifth Annual IEEE Symp. on Logic in Computer Science*, pages 414–425. IEEE Press, 1990.

[2] R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.

[3] J.W. Davies and S.A. Schneider. A Brief History of Timed CSP. *TCS*, 138, 1995.

[4] H. Dierks. PLC-Automata: A New Class of Implementable Real-Time Automata. In M. Bertran and T. Rus, editors, *ARTS'97*, volume 1231 of *LNCS*, pages 111–125, Mallorca, Spain, May 1997. Springer.

[5] H. Dierks. *Specification and Verification of Polling Real-Time Systems.* PhD thesis, University of Oldenburg, July 1999.

[6] H. Dierks. Synthesizing Controllers from Real-Time Specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(1):33–43, 1999.

[7] H. Dierks, A. Fehnker, A. Mader, and F.W. Vaandrager. Operational and Logical Semantics for Polling Real-Time Systems. In Ravn and Rischel [16], pages 29–40.

[8] H. Dierks and J. Tapken. Tool-Supported Hierarchical Design of Distributed Real-Time Systems. In *Proceedings of the 10th EuroMicro Workshop on Real Time Systems*, pages 222–229. IEEE Computer Society, June 1998.

[9] M.R. Hansen and Zhou Chaochen. Duration Calculus: Logical Foundations. *Formal Aspects of Computing*, 9:283–330, 1997.

[10] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111:193–244, 1994.

[11] B. Krieg-Brückner, J. Peleska, E.-R. Olderog, D. Balzer, and A. Baer. UniForM — Universal Formal Methods Workbench. In U. Grote and G. Wolf, editors, *Statusseminar des BMBF Softwaretechnologie*, pages 357–378. BMBF, Berlin, March 1996.

[12] K.G. Larsen, P. Petterson, and Wang Yi. Uppaal in a nutshell. *Software Tools for Technology Transfer*, 1(1+2):134–152, December 1997.

[13] O. Maler and A. Pnueli. Timing Analysis of Asynchronous Circuits using Timed Automata. In *Proc. CHARME'95*, volume 987 of *LNCS*, pages 189–205. Springer, 1995.

[14] O. Maler and S. Yovine. Hardware Timing Verification using Kronos. In *Proc. 7th Conf. on Computer-based Systems and Software Engineering*. IEEE Press, 1996.

[15] X. Nicollin, J. Sifakis, and S. Yovine. Compiling Real-Time Specifications into Extended Automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, September 1992.

[16] A.P. Ravn and H. Rischel, editors. *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1486 of *LNCS*, Lyngby, Denmark, September 1998. Springer.

[17] S.A. Schneider. An Operational Semantics for Timed CSP. *Information and Computation*, 116:193–213, 1995.

[18] J. Tapken and H. Dierks. MOBY/PLC – Graphical Development of PLC-Automata. In Ravn and Rischel [16], pages 311–314.

[19] S. Yovine. Kronos: a verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1+2):123–133, December 1997.

[20] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *IPL*, 40/5:269–276, 1991.

A Timed Automata

Timed automata are an automaton-based mathematical model for real-time systems. Although the basic concepts are very similar, various definitions of syntax and semantics can be found in the literature [1, 15, 2, 10, 13, 14]. Here we use a variant of timed automata that is defined in [14]:

Definition 7 (Timed Automaton). *A* timed automaton $\mathcal{T}$ *is a structure* $(\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{IV}, \mathcal{P}, \mu, S_0)$ *where:*

- $\mathcal{S}$ *is a finite set of* locations*,*
- $\mathcal{X}$ *is a finite set of real-valued variables called* clocks *whose values increase uniformly with time,*
- $\mathcal{L}$ *is a finite set of* labels*.*
- $\mathcal{E}$ *is a finite set of* edges *of the form* $e = (s, L, \phi, \rho, s')$*, or alternatively written as* $s \xrightarrow{L,\phi,\rho} s'$*, where* $s, s' \in \mathcal{S}$*,* $L \in \mathcal{L}$*,* ϕ *is a* clock constraint*, generated by the grammar* $\phi ::= x + c \leq d \mid c \leq x + d \mid x + c \leq y + d \mid \neg\phi \mid \phi_1 \wedge \phi_2$ *with* $x, y \in \mathcal{X}$ *and* $c, d \in \mathbb{R}$*, and* $\rho \subseteq \mathcal{X}$ *is the set of clocks which are to be reset to* 0 *by the transition,*
- $\mathcal{IV}$ *assigns to each location a clock constraint that serves as an* invariant *within the location,*
- $\mathcal{P}$ *is a finite set of atomic propositions,*
- μ *is a labelling of the locations with a set of atomic propositions over* $\mathcal{P}$*,*
- $S_0 \subseteq \mathcal{S}$ *is the set of* initial locations*.*

Usually only natural numbers are allowed as constants in the clock constraints, but in order to associate a timed automaton to each PLC-Automaton our definition allows for real-valued constants. The price we have to pay is that we cannot model-check this kind of timed automata. However, as long as the PLC-Automaton uses only discrete delays and a discrete cycle time, the corresponding timed automaton semantics uses only discrete time constants, too.

Definition 8 (Run of a timed automaton). *A* run *of* $\mathcal{T}$ *is an infinite sequence* $r = ((s_i, v_i, t_i))_{i \in \mathbb{N}_0}$ *where, for each* $i \in \mathbb{N}_0$*,* $s_i \in \mathcal{S}$ *is a location,* $v_i \in \mathcal{X} \longrightarrow \mathbb{R}_{\geq 0}$ *is a* valuation *of the clocks,* $t_i \in \mathbb{R}_{\geq 0}$ *is a time stamp, and* r *satisfies the following properties:*

- *the initial location is contained in* S_0*:* $s_0 \in S_0$*, initially all the clocks have value* 0*:* $\forall x \in \mathcal{X} : v_0(x) = 0$*, time starts at* 0*:* $t_0 = 0$*,*
- *the sequence of time stamps is monotonic and diverging:* $t_i \leq t_{i+1}$*, for all* $i \in \mathbb{N}_0$*, and* $\lim_{i \longrightarrow \infty} t_i = \infty$*,*
- *for all* $i \in \mathbb{N}_0$ *the invariant* $\mathcal{IV}(s_i)$ *is fulfilled during* $[t_i, t_{i+1}]$*:* $\forall 0 \leq t \leq t_{i+1} - t_i : \mathcal{IV}(s_i)(v_i + t)$ *with* $(v_i + t)(x) \overset{df}{=} v_i(x) + t$ *for all* $x \in \mathcal{X}$ *and* $\mathcal{IV}(s)(v)$ *denoting the evaluation of the constraint* $\mathcal{IV}(s)$ *at valuation* v*,*
- *for all* $i \in \mathbb{N}_0$ *there is an edge* $e = (s_i, L, \phi, \rho, s_{i+1})$ *such that*
  - *clock constraint* ϕ *holds at time* t_{i+1}*:* $\phi(v_i + t_{i+1} - t_i)$*, and*
  - *valuation* v_{i+1} *is updated according to* ρ*:* $\forall x \in \mathcal{X} : v_{i+1}(x) = 0$ *if* $x \in \rho$ *and* $v_i(x) + t_{i+1} - t_i$ *if* $x \notin \rho$

By $\mathcal{R}(\mathcal{T})$ *we denote the set of runs of a timed automaton* $\mathcal{T}$*.*

Definition 9 (Simulation, bisimulation). *Let $\mathcal{T}_i$ be Timed Automata with $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{X}_i, \mathcal{L}, \mathcal{E}_i, \mathcal{IV}_i, \mathcal{P}_i, \mu_i, S_{0,i})$ for $i = 1, 2$ and let β be a function of type $\mu_1(\mathcal{S}_1) \longrightarrow \mu_2(\mathcal{S}_2)$. We say that $\mathcal{T}_2$ is a* simulation *of $\mathcal{T}_1$ with respect to β (in symbols: $\mathcal{T}_1 \lesssim_\beta \mathcal{T}_2$) iff holds:*

$$\begin{aligned}&\forall((s^1_j, v^1_j, t_j)_{j\in\mathbb{N}_0}) \in \mathcal{R}(\mathcal{T}_1) :\\ &\quad \exists((s^2_j, v^2_j, t_j)_{j\in\mathbb{N}_0}) \in \mathcal{R}(\mathcal{T}_2) : \forall j \in \mathbb{N}_0 : \beta(\mu_1(s^1_j)) = \mu_2(s^2_j)\end{aligned}$$

We say that $\mathcal{T}_1$ is a bisimulation *of $\mathcal{T}_2$ with respect to β (in symbols: $\mathcal{T}_1 \approx_\beta \mathcal{T}_2$) iff β is a bijection, $\mathcal{T}_1 \lesssim_\beta \mathcal{T}_2$ and $\mathcal{T}_2 \lesssim_{\beta^{-1}} \mathcal{T}_1$.*