Operational and Logical Semantics for Polling Real-Time Systems*

Henning Dierks[2,**], Ansgar Fehnker[1,***], Angelika Mader[1,†], and Frits Vaandrager[1]

[1] Computing Science Institute, University of Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, the Netherlands
[2] University of Oldenburg, Germany

Abstract. PLC-Automata are a class of real-time automata suitable to describe the behavior of polling real-time systems. PLC-Automata can be compiled to source code for PLCs, a hardware widely used in industry to control processes. Also, PLC-Automata have been equipped with a logical and operational semantics, using Duration Calculus (DC) and Timed Automata (TA), respectively.
The three main results of this paper are: (1) A simplified operational semantics. (2) A minor extension of the logical semantics, and a proof that this semantics is *complete* relative to our operational semantics. This means that if an observable satisfies all formulas of the DC semantics, then it can also be generated by the TA semantics. (3) A proof that the logical semantics is *sound* relative to our operational semantics. This means that each observable that is accepted by the TA semantics constitutes a model for all formulas of the DC semantics.

1 Introduction

Programmable Logic Controllers (PLCs) are widely used in industry to control real-time embedded applications such as railway crossings, elevators, and production lines. PLCs are hardware devices that operate according to the simple but powerful architectural model of *polling real-time systems*. A polling real-time system behaves in cycles that can be split into three parts: the input values are polled, a new local state and output values are computed from the inputs and the old local state, and finally the new output values are written to the output ports. Depending for instance on the length of the computation, the duration of

** Supported by the German Ministery for Education and Research (BMBF), project UniForM, grant No. FKZ 01 IS 521 B3
*** Research supported by Netherlands Organization for Scientific Research (NWO) under contract SION 612-14-004
† Supported by the HCM Network EXPRESS and the Deutsche Akademischer Austauschdienst

a cycle may vary, but some upper bound ϵ on the cycle time is assumed to be available.

In this paper we study the operational and denotational semantics of polling real-time systems, i.e., the relationships between the input and output signals of such systems that are induced when a program is executed in real-time. Our work builds on recent work within the UniForM-project [11] on PLC-Automata [5–7,9]. PLC-Automata, basically an extension of classical Moore machines [10], can be viewed as a simple programming language for PLCs. In [5], a compilation scheme is given that generates runnable PLC-code for any given PLC-Automaton. Moreover, a logical (denotational) and an operational semantics of PLC-Automata are presented employing Duration Calculus (DC) [15,14] and Timed Automata (TA) [1], respectively. However, in [5] the relationships between these semantics are not further investigated.

The three main results established in this paper are:

1. A simplified operational semantics for PLC-Automata based on Timed Automata.
2. A minor extension of the logical semantics with some additional formulas, and a proof that this (restricted) semantics is *complete* relative to our operational semantics. This means that if an observable satisfies all formulas of the DC semantics, then it can also be generated by the TA semantics.
3. A proof that the logical semantics is *sound* relative to our operational semantics. This means that each observable that is accepted by the TA semantics constitutes a model for all formulas of the (extended) DC semantics.

An advantage of our operational semantics is that it is very intuitive, and provides a simple explanation of what happens when a PLC-Automaton runs on PLC hardware. Clearly, the 8 rules of the operational semantics are easier to understand than the 26 formulas of the DC semantics, especially for readers who are not experts in duration calculus. The operational semantics can also serve as a basis for automatic verification, using tools for timed automata such as KRONOS [4] and UPPAAL [3]. Our timed automata semantics uses 3 clock variables, which makes it more tractable for such tools than the semantics of [5] which requires 2 clocks plus one clock for each input value.

The logical semantics also has several advantages. Rather than modelling the internal state variables and hidden events of PLC hardware, it describes the allowed *observable* behaviour on the input and output ports. Duration Calculus, an interval temporal logic for real-time, constitutes a very powerful and abstract specification language for polling real-time systems. Via the DC semantics, proving that a PLC-Automaton $\mathcal{A}$ satisfies a DC specification *SPEC* reduces to proving that the duration calculus semantics $[\![\mathcal{A}]\!]_{\mathsf{DC}}$ logically implies *SPEC*. For this task all the proof rules and logical machinery of DC can be used. In fact, in [6] an algorithm is presented that synthesizes a PLC-Automaton from an (almost) arbitrary set of *DC Implementables*, a subset of the Duration Calculus that has been introduced in [13] as a stepping stone for specifying distributed real-time systems. In [13] a fully developed theory can be found how Implementables can

be obtained from general DC formulas. Hence, the synthesis algorithm provides a powerful means to design correct systems starting from specifications.

The fact that the TA and DC semantics are so different makes the proof of their equivalence interesting but also quite involved. In order to get the completeness result we had to extend the original DC semantics of [5] with 9 additional formulas of two lines each. These additional formulas, which express the presence of certain causalities between events, are not required for the correctness proof of the synthesis algorithm. This indicates that they may not be so important in applications. Nevertheless, we believe that the formulas do express fundamental properties of polling real-time systems and it is not so difficult to come up with an example of a situation in which the additional laws *are* used.

In this paper we only discuss the semantics of the simple PLC-Automata introduced in [5]. Meanwhile, PLC-Automata have been extended with a state charts like concept of hierarchy, in order allow for their use in the specification of complex systems [9]. We claim that it is possible to generalize the results of this paper to this larger class of hierarchical PLC-Automata. An interesting topic for future research is to give a low level operational semantics of PLCs, including hybrid aspects, clock drift, etc, and to prove that this low level semantics is a refinement of the semantics presented in this paper. Such a result would further increase confidence in the correctness of our semantic model.

2 PLC-Automata

In the UniForM-project [11] an automaton-like notion — called PLC-Automata — of polling real-time systems has been developed to enable formal verification of PLC-programs. Basically, Programmable Logic Controllers (PLCs), the hardware aim of the project, are just simple computers with a special real-time operating system. They have features for making the design of time- and safety-critical systems easier:

- PLCs have input and output channels where sensors and actuators, resp., can be plugged in.
- They behave in a cyclic manner where every cycle consists of three phases:
  - Poll all inputs and store the read values.
  - Compute the new values for the outputs.
  - Update all outputs.
- There is an upper time bound for a cycle, which depends on the program and on the number of inputs and outputs, that can be used to calculate an upper time bound for the reaction time.
- Convenient standardized libraries are given to simplify the handling of time.

The following formal definition of a PLC-Automaton incorporates the upper time bound for a polling cycle and the possibility of delay reactions of the system depending on state and input.

Definition 1. A *PLC-Automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega)$, where

- Q is a nonempty, finite set of *states*,
- Σ is a nonempty, finite set of *inputs*,
- δ is a function of type $Q \times \Sigma \longrightarrow Q$ (the *transition function*),
- $q_0 \in Q$ is the *initial state*,
- $\varepsilon \in \mathbb{R}_{>0}$ is the *upper bound* for a cycle,
- S_t is a function of type $Q \longrightarrow \mathbb{R}_{\geq 0}$ that tells for each state q how long the inputs contained in $S_e(q)$ should be ignored (the *delay time*),
- S_e is a function of type $Q \longrightarrow 2^{\Sigma}$ that gives for each state q the set of *delayed inputs*, i.e., inputs that cause no state transition during the first $S_t(q)$ time units after arrival in q,
- Ω is a nonempty, finite set of *outputs*,
- ω is a function of type $Q \longrightarrow \Omega$ (the *output function*)

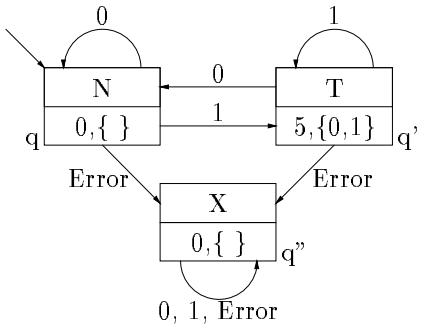
We require that the two following conditions hold, for all $q \in Q$ and $a \in \Sigma$,

$$S_t(q) > 0 \wedge a \notin S_e(q) \Longrightarrow \delta(q, a) \neq q \tag{1}$$

$$S_t(q) > 0 \Longrightarrow S_t(q) > \varepsilon \tag{2}$$

Restriction (1) is technical and needed to ensure the correctness of the PLC-source-code representing a PLC-Automaton w.r.t. the semantics given in [5]. It can be trivially met by adding, for each q, all actions a with $\delta(q, a) = q$ to the set $S_e(q)$. Restriction (2) says that delay times are either 0 or larger than the cycle upper bound time ϵ.

In the picture an example of a PLC-Automaton is given. A box representing a state (e.g. q') is annotated with the output (e.g. $\omega(q') = T$) in the upper part of the box and the pair of the delay time and the delay set in the lower part of the box (e.g. $S_t(q') = 5$, $S_e(q') = \{0, 1\}$). The system starts in state q with output N and remains in this state as long as the polled input is 0. The first time the polled input is not 0 the system changes state according to the transition function. If, following a 1-input, state q' is entered then the system will start a timer. Now the following cases are possible.



- The polled input is 0. In this case the system checks the timer. Only if the timer says that $S_t(q') = 5$ time units are over, the system takes the 0-transition back to state q. Otherwise the system stays in q'.
- The polled input is 1. In this case the system remains in q' independently from the status of the timer due to the fact that the 1-transition leads to q' again.
- The polled input is *Error*. In this case the system takes the *Error*-transition independently from the status of the timer because $\mathit{Error} \notin S_e(q')$.

We would like to stress that the range of applicability of PLC-Automata is much wider than just PLCs. In fact, PLC-Automata are an abstract representation of a machine that periodically polls inputs and has the possibility of measuring time.

3 The Timed Automaton Semantics of PLC-Automata

In this section we give an operational semantics of PLC-automata in terms of timed automata. For the definition of timed automata the reader is referred to [12,8]. We first present the components of the timed automaton $\mathcal{T}(\mathcal{A})$ that is associated to a given PLC-Automaton $\mathcal{A}$, and later give some intuition.

Each location[1] of $\mathcal{T}(\mathcal{A})$ is a 4-tuple (i, a, b, q), where $i \in \{0, 1, 2, 3\}$ describes the internal status of the PLC ("program counter"), $a \in \Sigma$ contains the current input, $b \in \Sigma$ contains the last input that was polled, and $q \in Q$ is the current state of the PLC-Automaton. There are three clocks in use: x measures the time that the latest input is stable, y measures the time spent in the current state, and z measures the time elapsed in the current cycle. The transitions of the timed automaton are defined in Table 1.

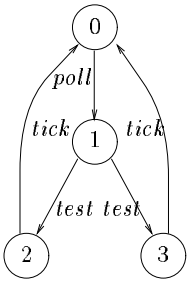
(i, a, b, q)	$\xrightarrow{c,\mathsf{true},\{x\}}$	(i, c, b, q)	if $c \neq a$	(ta-1)
$(0, a, b, q)$	$\xrightarrow{poll, 0<x \wedge 0<z, \emptyset}$	$(1, a, a, q)$		(ta-2)
$(1, a, b, q)$	$\xrightarrow{test, y \leq S_t(q), \emptyset}$	$(2, a, b, q)$	if $S_t(q) > 0 \wedge b \in S_e(q)$	(ta-3)
$(1, a, b, q)$	$\xrightarrow{test, y > S_t(q), \emptyset}$	$(3, a, b, q)$	if $S_t(q) > 0 \wedge b \in S_e(q)$	(ta-4)
$(1, a, b, q)$	$\xrightarrow{test,\mathsf{true},\emptyset}$	$(3, a, b, q)$	if $S_t(q) = 0 \vee b \notin S_e(q)$	(ta-5)
$(2, a, b, q)$	$\xrightarrow{tick,\mathsf{true},\{z\}}$	$(0, a, b, q)$		(ta-6)
$(3, a, b, q)$	$\xrightarrow{tick,\mathsf{true},\{z\}}$	$(0, a, b, q)$	if $q = \delta(q, b)$	(ta-7)
$(3, a, b, q)$	$\xrightarrow{tick,\mathsf{true},\{y,z\}}$	$(0, a, b, \delta(q, b))$	if $q \neq \delta(q, b)$	(ta-8)

Table 1: Transitions of timed automaton $\mathcal{T}(\mathcal{A})$

We now give some intuition about our semantics. Within the timed automaton we model the cyclic behaviour of a polling system. The events within one cycle are: polling input, testing whether input has to be ignored, producing new output (if necessary), and ending the cycle. The "program counter" models the phases of a cycle. The picture below shows how these events change the values of the program counter.

[1] Note that "locations" refer to the timed automaton and "states" to the PLC-Automaton.

- "0" denotes the first part of the cycle. The input has not yet been polled.
- "1" denotes that the polling has happened in the current cycle. The test whether to react has not been performed yet.
- "2" denotes that polling and testing have happened. The system decided to ignore the input.
- "3" denotes that polling and testing have happened. The system decided to react to the input.



A clock z is introduced within the timed automaton to measure the time that has elapsed within the current cycle. This clock is not allowed to go above the time upper bound ε and is reset at the end of each cycle.

In the first phase of a cycle incoming input is polled. In the timed automaton model there are no continuous variables available by which we can model continuous input. We have to restrict to a finite set Σ of inputs, and introduce for each $a \in \Sigma$ a transition label a that models the discrete, instantaneous event which occurs whenever the input signal changes value and becomes a. In principle, the input signal may change value at any time, not only during the first cycle phase. The current value of the input is recorded in the second component of a location of the timed automaton. Within our semantic model the occurrence of input events is described by transition (ta-1) (see Table 1).

The timed automaton model allows inputs that last only for one point of time, i.e., it is not required that time passes in between input events. However, it is of course realistic to assume that polling input takes some (small) amount of time, and that an input can only be polled if persists for some positive amount of time. Technically, we require that input may only be polled, if it has remained unchanged throughout a left-open interval. In the semantics we model this by a clock x that is reset whenever an input event occurs. Polling is only allowed if x is non-zero (see transition (ta-2)). The polled input is recorded in the third component of a location of $\mathcal{T}(\mathcal{A})$.

Next, we have to deal with input delay. Whether input has to be ignored or not depends on the state of the PLC-automaton and on the time during which the system has been in this state. The state of the PLC-automaton is recorded in the fourth component of the locations of $\mathcal{T}(\mathcal{A})$. Furthermore, we introduce a clock y that measures how long the current state is valid. Transitions (ta-3), (ta-4) and (ta-5) describe the cases that the *test*-event has to distinguish: if a state requires no delay or if it requires delay but the delay time is over, then input is not ignored and in the subsequent location the program counter has value 3; otherwise the program counter is assigned value 2.

A cycle ends with an event *tick* (see (ta-6), (ta-7), (ta-8)). If the program counter has value 3 then the PLC-Automaton state is updated in the new location according to the function δ; otherwise the PLC-Automaton state remains unchanged. After a *tick*-event the program counter gets value 0 and clock z is

reset to indicate that a new cycle has started. If the state changes then also clock y is reset. Formally, the timed automaton $\mathcal{T}(\mathcal{A})$ is defined as follows.

Definition 2. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega)$ be a PLC-Automaton. We define $\mathcal{T}(\mathcal{A}) \stackrel{\text{df}}{=} (\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I}, \mathcal{P}, \mu, S_0)$ with

- $\mathcal{S} \stackrel{\text{df}}{=} \{0,1,2,3\} \times \Sigma \times \Sigma \times Q$ as locations,
- $\mathcal{X} \stackrel{\text{df}}{=} \{x, y, z\}$ as clocks,
- $\mathcal{L} \stackrel{\text{df}}{=} \Sigma \cup \{poll, test, tick\}$ as labels,
- $\mathcal{I}(s) \stackrel{\text{df}}{=} z \leq \varepsilon$ as invariant for each location $s \in \mathcal{S}$,
- $\mathcal{P} = \Sigma \cup Q \cup \Omega$ as the set of propositions,
- $\mu(i, a, b, q) \stackrel{\text{df}}{=} a \wedge q \wedge \omega(q)$ as propositions for each location $(i, a, b, q) \in \mathcal{S}$,
- $S_0 \stackrel{\text{df}}{=} \{(0, a, b, q_0) | a, b \in \Sigma\}$ as set of initial locations,
- the set of transitions $\mathcal{E}$ consists of the transitions in Table 1, for each $i \in \{0,1,2,3\}$, $a, b, c \in \Sigma$, and $q \in Q$.

In [12,8], the operational behaviour of a timed automaton is defined in terms of accepting runs. An *accepting run* of a timed automaton $\mathcal{T} = (\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I}, \mathcal{P}, \mu, S_0)$ is an infinite sequence $r = ((s_i, v_i, t_i))_{i \in \mathbb{N}}$ where

- the $s_i \in \mathcal{S}$ are locations,
- the v_i are valuations of the clocks, i.e. $v_i \in [\mathcal{X} \longrightarrow \mathbb{R}_{\geq 0}]$,
- the $t_i \in \mathbb{R}_{\geq 0}$ form a diverging sequence of time stamps.

In order to be an accepting run, sequence r needs to satisfy a number of properties; we refer to [12,8] for the details. By $\mathcal{R}(\mathcal{T})$ we denote the set of accepting runs of a timed automaton $\mathcal{T}$.

4 The Duration Calculus Semantics of PLC-Automata

In this section we introduce logical semantics for PLC-Automata employing the Duration Calculus (see the Appendix for a brief introduction). It is based on the Duration Calculus semantics of [5].

We will give a set of formulae restricting the allowed interpretation of three observables $\mathsf{input} : \mathsf{Time} \longrightarrow \Sigma$, $\mathsf{state} : \mathsf{Time} \longrightarrow Q$, and $\mathsf{output} : \mathsf{Time} \longrightarrow \Omega$. We use q as abbreviation for $\mathsf{state} = q$ and assume that $q \in Q$. By $A, B, C \subseteq \Sigma$ we denote nonempty sets of inputs and each formula should be interpreted over all possible assignments to A, B, C.

The following set of formulae describe the general behavior of PLCs. Only inputs that arrived since the system switched into q may produce transitions (dc-1). Moreover, due to the cyclic behavior a transition can only be the consequence of an input which is not older than ε time units (dc-2).

$$\lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil \longrightarrow \lceil q \vee \delta(q, A) \rceil \qquad \text{(dc-1)}$$

$$\lceil q \wedge A \rceil \quad \xrightarrow{\varepsilon} \quad \lceil q \vee \delta(q, A) \rceil \qquad \text{(dc-2)}$$

We write A for $\mathsf{input} \in A$ and $\delta(q, A)$ for $\mathsf{state} \in \{q' \in Q \mid \exists a \in A : \delta(q, a) = q'\}$.

To ensure that the delay is observed we add a pair of formulae that corresponds to (dc-1) and (dc-2). They allow state changes only due to input not contained in $S_e(q)$.

$$S_t(q) > 0 \Longrightarrow \lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil \quad \xrightarrow{\leq S_t(q)} \quad \lceil q \vee \delta(q, A \setminus S_e(q)) \rceil \tag{dc-3}$$

$$S_t(q) > 0 \Longrightarrow \lceil \neg q \rceil \,;\, \lceil q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \quad \xrightarrow{\leq S_t(q)} \quad \lceil q \vee \delta(q, A \setminus S_e(q)) \rceil \tag{dc-4}$$

The formulae above do not force a transition, they only disallow some transitions. If we observe a change of the state, we know that another cycle will be finished within the next ε time units. In order to make this additional information exploitable we introduce formulae which force a reaction after at most ε time units depending on the valid input. First, two formulae for the case $S_t(q) = 0$:

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \longrightarrow \lceil \neg q \rceil \tag{dc-5}$$

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow$$
$$\lceil \neg q \rceil \,;\, (\lceil q \rceil^{>\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil) \longrightarrow \lceil \neg \delta(q, A) \setminus \delta(q, B) \rceil \tag{dc-6}$$

Formula (dc-5) covers the case that $q \notin \delta(q, A)$ holds on the whole interval of length ε right after the state change, whereas formula (dc-6) takes care of the case that input which satisfies $q \notin \delta(q, A)$, is followed by arbitrary input. The next two formulae apply to the case delay in state q. Note that (1) and $a \notin S_e(q)$ implies $q \neq \delta(q, a)$.

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow \lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \longrightarrow \lceil \neg q \rceil \tag{dc-7}$$

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow$$
$$\lceil \neg q \rceil \,;\, (\lceil q \rceil^{>\varepsilon} \wedge \lceil A \rceil \,;\, \lceil B \rceil) \quad \xrightarrow{\leq S_t(q)} \quad \lceil \neg \delta(q, A) \setminus \delta(q, B \setminus S_e(q)) \rceil \tag{dc-8}$$

Figure 1 shows an impossible behavior of the automaton on page 2 according to the TA semantics. We know that at t_0 and t_3 a cycle ends. Since $t_3 - t_0 > \varepsilon$ holds, the interval $[t_0, t_3]$ must at least contain two cycles. The first cycle produces no state change, therefore input 0 has been polled, input from interval $[t_1, t_3]$. Consequently, input 0 has also to be polled in the successive cycles. This however implies that the change to state T at t_3 can not happen. In the DC semantics formula (dc-6) applies to this situation. There is no delay in state N, and the only element of set A is 1.

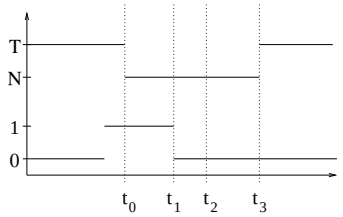


Fig. 1. Assume $t_1 - t_0 \leq \varepsilon$, $t_3 - t_1 \leq \varepsilon$ and $t_3 - t_0 > \varepsilon$

If the state changes we know that a cycle begins and will be completed within the next ε time units. The previous four DC formulae reflect that we have either two types of input or only one and there is a delay in current state or not. We have a set of formulae similar to (dc-5)-(dc-8) concerning intervals of length 2ε with a stable state. These intervals must also contain at least one cycle. However, for this situation we do not only have to consider the cases *no delay* and *delay active* but also the cases *delay has expired* and *delay expires* . If there is no delay

active in state q then the following two formulae apply.

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow \Box(\lceil q \wedge A \rceil \Longrightarrow \ell < 2\varepsilon) \tag{dc-9}$$

$$S_t(q) = 0 \wedge q \notin \delta(q, A) \Longrightarrow$$
$$(\lceil q \rceil \wedge \lceil A \rceil^{>\varepsilon} ; \lceil B \rceil) \xrightarrow{2\varepsilon} \lceil \neg\delta(q, A) \setminus \delta(q, B) \rceil \tag{dc-10}$$

In the case that $S_t(q) > 0$ holds and the delay time has not expired only inputs not contained in $S_e(q)$ can force a transition to happen.

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow \Box(\lceil q \wedge A \rceil \Longrightarrow \ell < 2\varepsilon) \tag{dc-11}$$

$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \Longrightarrow$$
$$\lceil \neg q \rceil ; \lceil q \rceil ; (\lceil q \rceil^{2\varepsilon} \wedge \lceil A \rceil^{>\varepsilon} ; \lceil B \rceil) \xrightarrow{\leq S_t(q)} \lceil \neg\delta(q, A) \setminus \delta(q, B \setminus S_e(q)) \rceil \tag{dc-12}$$

If the delay time is expired the system behaves like a system with no delay. The following formulae are consequently quite the same as (dc-9) and (dc-10), except that the state has to be stable for an additional $S_t(q)$ time units.

$$S_t(q) > 0 \wedge q \notin \delta(q, A) \Longrightarrow \Box(\lceil q \rceil^{S_t(q)} ; \lceil q \wedge A \rceil \Longrightarrow \ell < S_t(q) + 2\varepsilon) \tag{dc-13}$$

$$S_t(q) > 0 \wedge q \notin \delta(q, A) \Longrightarrow$$
$$\lceil q \rceil^{S_t(q)} ; (\lceil q \rceil \wedge \lceil A \rceil^{>\varepsilon} ; \lceil B \rceil) \xrightarrow{S_t(q)+2\varepsilon} \lceil \neg\delta(q, A) \setminus \delta(q, B) \rceil \tag{dc-14}$$

To express that the delay time expires during an interval we need some more complicated formulae, but the idea is the same as in the foregoing cases.

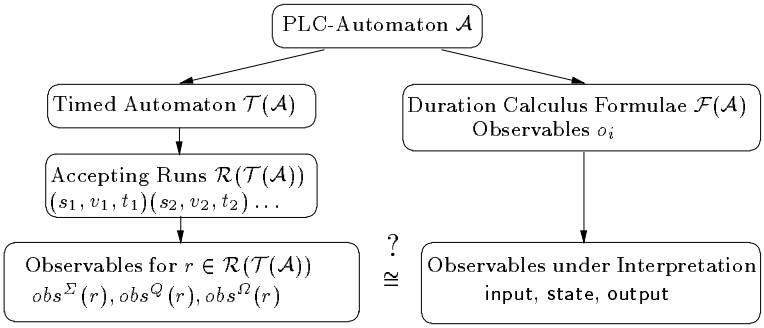
$$S_t(q) > 0 \wedge q \notin \delta(q, B) \wedge A \cap S_e(q) = \emptyset \Longrightarrow$$
$$\Box(\lceil q \rceil \wedge (l = t_1); \lceil A \rceil^{t_2} ; \lceil B \rceil^{t_3} \wedge t_1 + t_2 = S_t(q) \Longrightarrow t_2 + t_3 < 2\varepsilon) \tag{dc-15}$$

$$S_t(q) > 0 \wedge q \notin \delta(q, B) \wedge A \cap S_e(q) = \emptyset \Longrightarrow$$
$$\lceil \neg q \rceil ; \left(\lceil q \rceil \wedge (l = t_1); \lceil A \rceil^{t_2} ; \lceil B \rceil^{t_3} ; \lceil C \rceil^{t_4} \begin{array}{ll} \wedge & t_2 + t_3 + t_4 = 2\varepsilon \\ \wedge & t_1 + t_2 = S_t(q) \\ \wedge & t_2 + t_3 > \varepsilon \end{array} \right)$$
$$\longrightarrow \lceil \neg(\delta(q, A \cup B) \setminus \delta(q, C)) \rceil \tag{dc-16}$$

The DC-semantics presented in this paper differs from the one presented in [5] by the additional formulae (dc-6), (dc-8), (dc-10), (dc-12), (dc-14), (dc-15), (dc-16). We also used $<$ instead of $\leq$ in the formulae (dc-9), (dc-11) and (dc-13). We did not mention two formulae that define the initial state and relate output to states. Those can be found together with the original semantics in [5,8]. Some of the formulae given in this section are not applicable in the initial phase. The corresponding formulae for the initial phase can be found in the full paper [8].

5 Observables and Runs

In this section we establish the relationship between Duration Calculus and timed automata. The semantic objects of the Duration Calculus are interpretations of observables; the semantic objects of timed automata are accepting runs. We have to define suitable mappings between these semantic objects and show equi-expressiveness of both semantics.



In order to compare timed automata and Duration Calculus within our setting, we need to relate sets of atomic propositions to observables. A sufficient condition for doing this is that, for each observable *obs* with domain P, elements of P are propositions of the timed automaton and P is a *league*.

Definition 3. For a timed automaton $\mathcal{T} = (\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I}, \mathcal{P}, \mu, S_0)$, we define a set $P \subseteq \mathcal{P}$ of propositions to be a *league*, if at each location one and only one proposition $p \in P$ is valid, i.e., $\forall s \in S\ \exists^1 p \in P : p \in \mu(s)$.

For $\mathcal{T}(\mathcal{A})$ as in Definition 2 there are three leagues corresponding to the three observables in the DC semantics $\mathcal{F}(\mathcal{A})$: Σ corresponds to input, Q to state, and Ω to output.

Recall that the interpretation of a DC observable is a function from time to the domain of this observable. In the case of accepting runs of a timed automaton time needs not to increase strictly monotonically. Therefore, when mapping an accepting run to an observable, we have to associate to each point of time a unique element of the domain, i.e., a proposition of a league. If in an accepting run there are consecutive states at the same point in time, we choose the last one as the unique interpretation.

Definition 4. Let $\mathcal{T}(\mathcal{A})$ be a timed automaton, $r = ((s_i, v_i, t_i))_{i \in \mathbb{N}}$ an accepting run of $\mathcal{T}(\mathcal{A})$, and $P \subseteq \mathcal{P}$ a league. We define obs^P as a function from runs to observables as follows:

$$obs^P \stackrel{\text{df}}{=} \begin{cases} \mathcal{R}(\mathcal{T}(\mathcal{A})) \longrightarrow (\mathsf{Time} \longrightarrow P) \\ r \quad \mapsto \quad (t \quad \mapsto\ p \text{ where } p \in \mu(s_i) \text{ and } i = \max\{i \mid t_i \leq t\}). \end{cases}$$

Note that $obs^P(r)$ is finitely variable due to the divergence of time in the accepting run r.

In DC the truth of a formula is defined by integrals of functions over intervals. Functions that are identical up to zero-sets give the same truth values for all formulae and can be identified. We define two interpretations obs_I and $obs_{I'}$ of observable *obs* to be *equivalent*, notation $obs_I \cong obs_{I'}$, if the interpretations differ in at most countably many points. Hence, by definition of $\cong$ we have that, for each Duration Calculus formula F,

$$obs_1 \cong obs_2 \Longrightarrow (F \text{ holds for } obs_1 \iff F \text{ holds for } obs_2).$$

Definition 5. Let $\{o_1, \ldots, o_n\}$ be a set of observables with disjoint domains $D_1, \ldots, D_n$, $\mathcal{F}$ a set of DC formulae, and $Int_{\mathcal{F}}$ the set of interpretations of

$o_1, \ldots, o_n$ that satisfy the formulae of $\mathcal{F}$. Let $\mathcal{T} = (\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I}, \mathcal{P}, \mu, S_0)$ be a timed automaton with leagues $D_i \subseteq \mathcal{P}$ for $1 \leq i \leq n$. We say that $\mathcal{T}$ is

- *sound* w.r.t. $\mathcal{F}$ if for all accepting runs $r \in \mathcal{R}(\mathcal{T})$ there exists an interpretation $I \in Int_{\mathcal{F}}$ such that for $i = 1, \ldots, n$ it is $o_{i,I} \cong obs^{D_i}(r)$;
- *complete* w.r.t. $\mathcal{F}$ if for each interpretation $I \in Int_{\mathcal{F}}$ there exists an accepting run $r \in \mathcal{R}(\mathcal{T})$ such that for $i = 1, \ldots, n$ it is $o_{i,I} \cong obs^{D_i}(r)$.

The following theorem, which is the main result of this paper, states that the operational semantics $\mathcal{T}(\mathcal{A})$ is both sound and complete w.r.t. the logical semantics $\mathcal{F}(\mathcal{A})$. For the proof of this result we refer to [8].

Theorem 6. Let $\mathcal{A}$ be a PLC-Automaton. Then $\mathcal{T}(\mathcal{A})$ is sound and complete with respect to $\mathcal{F}(\mathcal{A})$, i.e.,
(i) for each accepting run $r \in \mathcal{R}(\mathcal{T}(\mathcal{A}))$ there exists an interpretation I of the observables input, state and output that satisfies all formulae of $\mathcal{F}(\mathcal{A})$, such that $\mathsf{input}_I \cong obs^{\Sigma}(r)$, $\mathsf{state}_I \cong obs^{Q}(r)$ and $\mathsf{output}_I \cong obs^{\Omega}(r)$;
(ii) for each interpretation I of the observables input, state and output that satisfies all formulae of $\mathcal{F}(\mathcal{A})$, there exists an accepting run $r \in \mathcal{R}(\mathcal{T}(\mathcal{A}))$ such that $\mathsf{input}_I \cong obs^{\Sigma}(r)$, $\mathsf{state}_I \cong obs^{Q}(r)$ and $\mathsf{output}_I \cong obs^{\Omega}(r)$.

References

1. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
3. J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL: a tool suite for the automatic verification of real-time systems. In Alur et al. [2], pages 232–243.
4. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Alur et al. [2], pages 208–219.
5. H. Dierks. PLC-Automata: A New Class of Implementable Real-Time Automata. In M. Bertran and T. Rus, editors, *ARTS'97*, volume 1231 of *Lecture Notes in Computer Science*, pages 111–125, Mallorca, Spain, May 1997. Springer-Verlag.
6. H. Dierks. Synthesising Controllers from Real-Time Specifications. In *Tenth International Symposium on System Synthesis*, pages 126–133. IEEE CS Press, September 1997.
7. H. Dierks and C. Dietz. Graphical Specification and Reasoning: Case Study "Generalized Railroad Crossing". In J. Fitzgerald, C.B. Jones, and P. Lucas, editors, *FME'97*, volume 1313 of *Lecture Notes in Computer Science*, pages 20–39, Graz, Austria, September 1997. Springer-Verlag.
8. H. Dierks, A. Fehnker, A. Mader, and F. Vaandrager. Operational and logical semantics for polling real-time systems. CSI-R9813, University of Nijmegen, april 1998.
9. H. Dierks and J. Tapken. Tool-Supported Hierarchical Design of Distributed Real-Time Systems. In *Proceedings of EuroMicro 98*, 1998. to appear.
10. Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, Inc., 1970.

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. O. Maler and S. Yovine. Hardware Timing Verification using Kronos. In *Proc. 7th Conf. on Computer-based Systems and Software Engineering*. IEEE Press, 1996.
13. A.P. Ravn. Design of Embedded Real-Time Computing Systems. Technical Report 1995-170, Technical University of Denmark, 1995.
14. Zhou Chaochen. Duration Calculi: An overview. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Formal Methods in Programming and Their Application*, volume 735 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1993.
15. Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Inform. Proc. Letters*, 40/5:269–276, 1991.

A Duration Calculus

For a proper introduction to the DC see e.g. [15,14].

Formulae of the DC are propositions about time-dependent variables *obs*, called *observables*. An interpretation I assigns to each observable *obs* a function $obs_I : \mathsf{Time} \longrightarrow D$, where $\mathsf{Time} = \mathbb{R}_{\geq 0}$ and D is the type of *obs*.

Terms are built over observables and have a certain type. Terms of Boolean type are called *state assertions*, e.g., $obs = v$ for $v \in D$. For an interpretation I they denote functions $P_I : \mathsf{Time} \longrightarrow \{0, 1\}$.

Duration terms are of type real and their values depend on a given time interval $[b, e]$. The simplest duration term is the symbol ℓ denoting the length $e - b$ of $[b, e]$. For each state assertion P there is a duration term $\int P$ measuring the duration of P. Semantically, $\int P$ denotes $\int_b^e P_I(t)dt$ on the interval $[b, e]$. Real-valued operators applied to duration terms are also duration terms.

Duration formulae are built from boolean-valued operations on duration terms, the special symbols true and false, and they are closed under propositional connectives, the chop-operator ";", and quantification over rigid variables. Their truth values depend on a given interval. We use F for a typical duration formula. Constants true and false evaluate to true resp. false on every given interval. The composite duration formula $F_1; F_2$ holds in $[b, e]$ if this interval can be divided into an initial subinterval $[b, m]$ where F_1 holds and a final subinterval $[m, e]$ where F_2 holds.
Besides this basic syntax various abbreviations are used: $\lceil\,\rceil \stackrel{\text{df}}{=} \ell = 0$ (point interval), $\lceil P \rceil \stackrel{\text{df}}{=} \int P = \ell \wedge \ell > 0$ (everywhere), $\Diamond F \stackrel{\text{df}}{=} \mathsf{true}; F; \mathsf{true}$ (somewhere), $\Box F \stackrel{\text{df}}{=} \neg\Diamond\neg F$ (always), $F^t \stackrel{\text{df}}{=} (F \wedge \ell = t)$, $F^{\sim t} \stackrel{\text{df}}{=} (F \wedge \ell \sim t)$, where $\sim \in \{<, \leq, >, \geq\}$.
A duration formula F *holds* in an interpretation I if F evaluates to true in I and every interval of the form $[0, t]$ with $t \in \mathsf{Time}$.

The following so-called *standard forms* are useful to describe dynamic behaviour: $F \longrightarrow \lceil P \rceil \stackrel{\text{df}}{=} \Box\neg(F; \lceil\neg P\rceil)$ (followed-by), $F \stackrel{t}{\longrightarrow} \lceil P \rceil \stackrel{\text{df}}{=} (F \wedge \ell = t) \longrightarrow \lceil P \rceil$ (timed leads-to), $F \stackrel{\leq t}{\longrightarrow} \lceil P \rceil \stackrel{\text{df}}{=} (F \wedge \ell \leq t) \longrightarrow \lceil P \rceil$ (timed up-to).